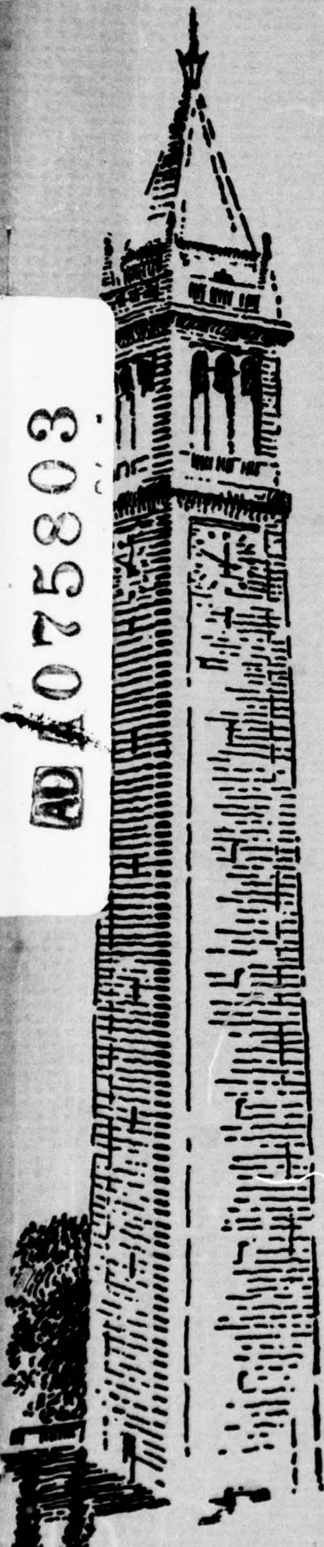
AD-A075 803 CALIFORNIA UNIV BERKELEY ELECTRONICS RESEARCH LAB F/G 9/2
SOLVER: AN ANALYTIC FUNCTION ROOT SOLVING AND PLOTTING PACKAGE.(U)
AUG 79 H S AU-YEUNG , A FRIEDMAN N00014-77-C-0578
UNCLASSIFIED UCB/ERL-M79/55 NL

| OF |

AD
A075803

END
DATE
FILMED
11-79
DDC



SOLVER: AN ANALYTIC FUNCTION ROOT SOLVING AND PLOTTING PACKAGE

by

H. Stephen Au-Yeung and Alex Friedman

Memorandum No. UCB/ERL M79/55

31 August 1979

ELECTRONICS RESEARCH LABORATORY

College of Engineering

University of California, Berkeley, CA 94720

(6) SOLVER: AN ANALYTIC FUNCTION ROOT SOLVING

AND PLOTTING PACKAGE.

by

(10) H. Stephen Au-Yeung [redacted] Alex Friedman

(12) 77

(9) repts

(14) Memorandum No. UCB/ERL-M79/55

(11) 31 Aug[redacted]979

(15) NØØØ14-77-C-Ø578, DOE-DE-ASØ3-76SFØØØ34

Accession For	
NTIS GRA&I	✓
DDC TAB	
Unannounced	
Justification Letter on file	
By	
Distribution/	
Availability Codes	
Dist.	Avail and/or special
A	

127 550

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

127 550

Contents

Acknowledgments

We wish to acknowledge the contributions of Y.J. Chen and D. Harned, who helped test the program and offered useful suggestions, and that of Professor C.K. Birdsall, whose comments helped make this document useful. This work was supported by DOE contract DE-AS03-76SF00034, Project Agreement DE-AT03-76ET53064 and ONR contract N00014-77-C-0578.

I. Introduction

SOLVER is a root solver that provides automatic compilation, loading, and execution. It uses a simple version of Muller's method ~~(see Ref. 1)~~. SOLVER is intended for the occasional user who needs roots but balks at learning to use sophisticated but complicated solvers, which usually also require considerable additional programming. The user need only supply the function to be set to zero and the number of solutions desired. SOLVER is available on the CRAY-1 computer and can be obtained by typing:

```
  rfilem  read 1222 .cray solver(LF)end  /  t v
```

This report corresponds to the SOLVER version of August 31, 1979. Later versions of SOLVER will be stored in FILEM directory .cray of user number 1222. The user should periodically check the date of this file (filem how 1222 .cray solver) to see if the program has been updated. The file SOLVER is a LIB file; it contains the latest sources as well as the latest documentation. To obtain the documentation, type:

```
  lib  solver(LF)x solv/doc(LF)end  /  t v
  netout  [usc] solv/doc box nnn solver  /  t v
```

A more robust but more complicated Muller subroutine is MRAF, CACM algorithm 196. Changes suggusted by J. Traub and the certification appeared in CACM January 1968 p.12. A FORTRAN translation was made and some remaining glitches were removed and new features added by A.B. Langdon, UC Berkeley Febrary 1968. This subroutine is used in the ROOTS program (see Ref. 2,3). If the demand is sufficient, this latter subroutine may be made available as an option at some future date.

Other root solvers are available on the MFECC CDC-7600; these might be transferred to the CRAY by a user. They include ABEROOT, by C.E. Seyler, which employs a global Newton method, and routines RPZERO, CPZERO for finding roots of real and complex polynomials, developed at LASL (see Ref. 5).

II. Elementary SOLVER

To run SOLVER, the user enters "solver" followed by some commands from the terminal. In this section, some basic commands are introduced:

<integer> -
: Specify the number of roots desired (defaults to 1 and cannot exceed 1024; this is referred to as n in the rest of this report).

f [file name] -
: Enter the function to be solved. If the function is to be entered from the terminal, the user should wait for the prompt ":" to appear. The name of the function is "f" and the argument is "z". The input format is assumed to be that of FORTRAN statements. In the tty-input mode, input is terminated when "f" is the first non-blank character followed by "=" or a blank, or a slash is the first non-blank character of an input line.

go [file name or "tty"] -
: Compile, load and execute. At execution time, a number of parameters can be set to override the default values. If tty is specified, the user should wait for the prompt "-" to appear. Inputs are in the format of a NAMELIST (terminated by $).
: Some of these parameters are:

```
n      - Number of roots desired (defaults to <integer>).
maxit  - Maximum number of iterations per root allowed
         (defaults to 100).
known  - Number of known roots (defaults to 0).
h      - Initial distance between estimated roots
         (defaults to 0.5).
ep1    - Relative error tolerance on z(i) (defaults to
         1.e-12 which is also the minimum one can specify).
ep2    - Error tolerance on f(z(i)) (defaults to 1.e-20
         which is also the minimum one can specify).
of     - = 0  => Output results to tty only (default).
           1  => Output results to tty and a disk file.
           2  => Output results to a disk file only.
           3  => Do not output results.
         See Section III for the name of disk file.
y      - Array for initial estimated values and result of
         roots (defaults to 0).
dbug   - = 0 => Drop file deleted after execution (default).
           1 => Drop file retained after execution.
```

xeq [file name or "tty"] -
: Execute the generated program again without recompiling it. See the command "go" for details.

savef <file name> -

Save the function into the file <file name> which can be read by using the command "f" later.

kbin -
Keep the binary library "solv/b/l" in the active file area. If SOLVER is being using often, this step conserves computer effort (the cost of extracting "solv/b/l" from the library file each time is eliminated).

end -
Terminate SOLVER.

Example II.1:
Most common syntax

```
         user:  solver / 2 2
routine      :  *** root solver  79.08  ***
routine      :  extracting solv/b/l
routine      :  c 07/20/79 12:39:32 001222
routine/user:  >3
routine/user:  >f
routine/user:  :       f = z**3 - 8.
routine/user:  >go
routine      :  compiling and loading
routine      :  FT004 - CFT VERSION -    07/14/79 SCHEDULER
routine      :  FT001 - COMPILE TIME =     0.0065 SECONDS
routine      :  *** cray loader version - c121  07/05/79
routine      :  executing
routine      :  roots:
routine      :    (   2.00000000000e+00 ,   0.00000000000e-01 )
routine      :    (  -1.00000000000e+00 ,   1.73205080757e+00 )
routine      :    (  -1.00000000000e+00 ,  -1.73205080757e+00 )
routine/user:  >end
routine      :    all done
```

Example II.2:
All commands on execute line

```
         user:  solver  20 f go end  /  2 3
routine      :  *** root solver  79.08  ***
routine      :  extracting solv/b/l
routine      :  c 07/20/79 12:39:32 001222
routine/user:  :       f = csin ( z )
routine      :  compiling and loading
routine      :  FT004 - CFT VERSION -    07/14/79 SCHEDULER
routine      :  FT001 - COMPILE TIME =     0.0061 SECONDS
routine      :  *** cray loader version - c121  07/05/79
routine      :  executing
routine      :  roots:
routine      :    (   0.00000000000e-01 ,   0.00000000000e-01 )
routine      :    (  -3.14159265359e+00 ,   9.33926735795e-21 )
routine      :    (   3.14159265359e+00 ,  -4.28706032686e-36 )
```

```
routine      :    (  -6.28318530718e+00 ,   1.55786371627e-25 )
routine      :    (   6.28318530718e+00 ,  -1.81515384110e-17 )
routine      :    (  -1.25663706144e+01 ,   6.94470731794e-23 )
routine      :    (   1.25663706144e+01 ,  -4.06743060262e-20 )
routine      :    (  -9.42477796077e+00 ,   3.36131166524e-27 )
routine      :    (   9.42477796077e+00 ,   4.33163059989e-19 )
routine      :    (  -2.51327412287e+01 ,   2.44708281560e-24 )
routine      :    (   2.51327412287e+01 ,   2.46849267576e-19 )
routine      :    (  -2.19911485751e+01 ,   2.31592585486e-23 )
routine      :    (   2.19911485751e+01 ,   3.54161507604e-17 )
routine      :    (  -3.45575191895e+01 ,  -9.21324826052e-27 )
routine      :    (   3.45575191895e+01 ,   7.42544852601e-19 )
routine      :    (  -1.88495559215e+01 ,  -4.60537034333e-22 )
routine      :    (   3.14159265359e+01 ,  -3.34489216701e-16 )
routine      :    (   1.88495559215e+01 ,  -1.30075028143e-26 )
routine      :    (  -4.71238898038e+01 ,   1.40250535984e-15 )
routine      :    (  -4.39822971503e+01 ,   5.50257927188e-20 )
routine      :   all done
```

If the user wants roots in some kind of order, and hopefully with none skipped, the user must provide proper initial guesses for the y values. It is also possible to ask the roots to be reordered by the user subroutine (see next section).

III. Intermediate SOLVER

It is sometimes desirable to find different sets of roots of a function with parameters assuming different values. SOLVER provides this capability by allowing the user to supply a subroutine. The structure of the main program consists of the following:

```
   40 continue
           call <the subroutine>
           if ( l(30).lt.0 )  call exit ( dbug )
           call <muller to find roots>
           write <results>
           go to 40
```

There are variables which are common to the main program, the function and the user-supplied subroutine for the user to use; they are:

```
        integer    l(30)
        complex    c(30), p(1024,100)
```

These variables can be read in either from the terminal or from the input file during execution. As the user may have noticed, l(30) is the control variable for looping to find roots. The loop stops when l(30) < 0.

It is possible to "follow" a root as parameters change by careful programming of the option subroutine, but the user must implement this him/herself.

Several I/O units are also available in the subroutine:

59 - Terminal (tty).
05 - Input (exists only if "go <file name>" or "xeq <file name>" is given - logical unit 5 is connected to the file <file name>).
06 - Output (file name is "rout" with the current suffix and a letter appended. Exists only if of=1 or 2).

Libraries such as DISSPLA, TV80LIB, FORTLIB and BASELIB are linked at load time automatically. However, the user may call subroutines from other libraries; SOLVER allows the user to add and/or delete libraries at load time.

The following are additional commands available in SOLVER:

sub [file name] -
  Enter the subroutine. All input format and termination rules are similar to the command "f" in Section II.

saves <file name> -
Save the subroutine into the file <file name> which can be read by using the command "sub" later.

alib [lib name ...] -
Add libraries at load time. This command is terminated by the carriage return key, the line feed key, or the escape key.

dlib [lib name ...] -
Delete libraries. The syntax of this command is the same as that of the command "alib".

<else> -
If the input is the name of an executable file, SOLVER will run it as a controllee. This is useful when using a text editor to modify the function, the subroutine or the input file, or using DDT to debug the program.
In this case, the rest of the input line is passed to the controllee, i.e. any command following will be lost to SOLVER.

Example III.1:

```
          user:  solver  /  2 2
routine       :  *** root solver  79.08  ***
routine       :  extracting solv/b/1
routine       :  c 07/20/79 12:39:32 001222
routine/user:  >3 sub
routine/user:  :       data      l(30) /3/, c(1) /64./
routine/user:  :c
routine/user:  :       l(30) = l(30) - 1
routine/user:  :       if ( l(30).lt.0 )  return
routine/user:  :       c(1) = c(1) / 2.
routine/user:  :       write(59,100)  c(1)
routine/user:  :  100 format ( "c(1) = ", 2f14.6 )
routine/user:  :/
routine/user:  >f go end
routine/user:  :       f = z**3 - c(1)
routine       :  compiling and loading
routine       :  FT004 - CFT VERSION -   07/14/79 SCHEDULER
routine       :  FT001 - COMPILE TIME =    0.0136 SECONDS
routine       :  *** cray loader version - c121  07/05/79
routine       :  executing
routine       :  c(1) =       32.000000       0.000000
routine       :  roots:
routine       :   (   3.17480210394e+00 ,   0.00000000000e-01 )
routine       :   (  -1.58740105197e+00 ,   2.74945927400e+00 )
routine       :   (  -1.58740105197e+00 ,  -2.74945927400e+00 )
routine       :  c(1) =       16.000000       0.000000
routine       :  roots:
routine       :   (   2.51984209979e+00 ,  -4.17757316097e-28 )
routine       :   (  -1.25992104989e+00 ,   2.18224727194e+00 )
```

```
routine      :  (  -1.25992104989e+00 ,  -2.18224727194e+00 )
routine      :  c(1) =        8.000000      0.000000
routine      :  roots:
routine      :  (   2.00000000000e+00 ,   2.44348158944e-46 )
routine      :  (  -1.00000000000e+00 ,   1.73205080757e+00 )
routine      :  (  -1.00000000000e+00 ,  -1.73205080757e+00 )
routine      :   all done
```

Example III.2:
Deleting the default graphic libraries which are not used because no plots are made.

```
         user:  solver  /  2 1
routine      :  *** root solver  79.08  ***
routine      :  extracting solv/b/1
routine      :  c 07/20/79 12:39:32 001222
routine/user:  >dlib  disspla tv80lib
routine/user:  >2 f go end
routine/user:  :       f = z**2 - 16.
routine      :  compiling and loading
routine      :  FT004 - CFT VERSION -   07/14/79 SCHEDULER
routine      :  FT001 - COMPILE TIME =    0.0063 SECONDS
routine      :  *** cray loader version - c121  07/05/79
routine      :  ***warning-unsatisfied externals***
routine      :  executing
routine      :  roots:
routine      :  (  -4.00000000000e+00 ,  -2.01948391737e-28 )
routine      :  (   4.00000000000e+00 ,   6.45718332361e-42 )
routine      :   all done
```

IV. Advanced SOLVER

This section introduces the plotting capability of SOLVER.

A. Plotting Roots vs. One Parameter

SOLVER can plot the roots of a function vs. a user defined parameter whose values are to be stored in x, a real array of size 1024. To do so, the user has to set the variable "plotmode" to 1, 2, or 3 depending upon the destination desired for the plot (see below) and must assign values to x in the subroutine (see below). Several parameters are included for the user to set at execution time; they are:

```
plotmode - = 0  =>  No plot (default).
             1  =>  Fr80 file.
             2  =>  Printer file (always called "disout").
             3  =>  Tektronix.
            If the value read for plotmode is positive, the user
            can vary the value of plotmode between 0 and a
            positive integer dynamically.  The user assigns a
            positive number to plotmode only for those roots
            he/she wants to plot.  The number of abscissa values
            plotted is the number of times the plotmode is found
            to be positive upon return from the subroutine.

plotid   - Depending on the destination, plotid has different
           interpretations.  If plotmode is
           1 (fr80 file)     => Name of this plot in not more than
                                8 characters.
           2 (printer file)  => Number of characters per line
                                (integer).
           3 (tektronix)     => The baud rate (integer).

id       - In less than 30 characters, this is the box and id.

mapset   - = 0  =>  A linear-linear plot (default).
             1  =>  A linear-log plot (x linear, y log).
             2  =>  A log-linear plot (y linear, x log).
             3  =>  A log-log plot.

xsize    - The length (in inches) of the x-axis (defaults to 6.0).
           DISSPLA conventions for "inches" are employed, i.e. the
           "page" is assumed to be 8.5x11 inches.

ysize    - The length (in inches) of the y-axis (defaults to 4.5).

grids    - = 0  =>  No grid (default).
             1  =>  Grid.

x        - The user can also input the values of x from tty or from
           the input file.
```

The program structure now looks as follows:

```
40 continue
       call <the subroutine>
       if ( not the first time and plotmode > 0 )  call <store y>
       if ( l(30).lt.0 )  go to 70
       call <muller to find roots>
       write <result>
       go to 40
70 continue
       if ( plot desired )  call <plot>
       call exit ( dbug )
```

The graphics library used is DISSPLA. All warning or error messages, if any, will be sent to the file named "disout". The maximum number of roots which can be plotted is 25; however, since DISSPLA has only 15 different symbols to distinguish different curves, the user is advised not to plot more than 15 roots.

The plotted output includes:

(1) The contents of the input file, if there is one.
(2) The real part of the roots vs. the parameter (x).
(3) The imaginary part of the roots vs. the parameter (x).

The size of a page is assumed to be 8.5"x11", regardless of output medium selected. If possible, SOLVER will place items (2) and (3) on one page. The maximum size of graphs to be plotted two-to-a-page is 6"x4.5" (or 4.5"x6" for side-by-side plots).

For an fr80 file, the user must do either of the following after finding out the name of the fr80 file:

(1) netplot [usc] <fr80 file> [1.] [turn.] / t v   or
(2) netout a <fr80 file> b. / t v
    (then log-off the CRAY-1 and log-on to the 7600 and view the plot file on a tektronix by FR80PLOT).

These operations (without the time and value) can be done on the same suffix under which SOLVER is running.

Example IV.A.1 (see Fig. IV.A.1):
A "back-door" way to plot sin(x) using SOLVER

```
          user:  solver
 routine      :  *** root solver  79.08  ***
 routine      :  extracting solv/b/l
 routine      :  c 07/22/79 20:19:41 001222
 routine/user:  >sub
 routine/user:  :       equivalence ( ixind,l(29) )
 routine/user:  :       data       l(30) /21/, ixind /0/
 routine/user:  :       data       pi /3.1415926535/
 routine/user:  :c
```

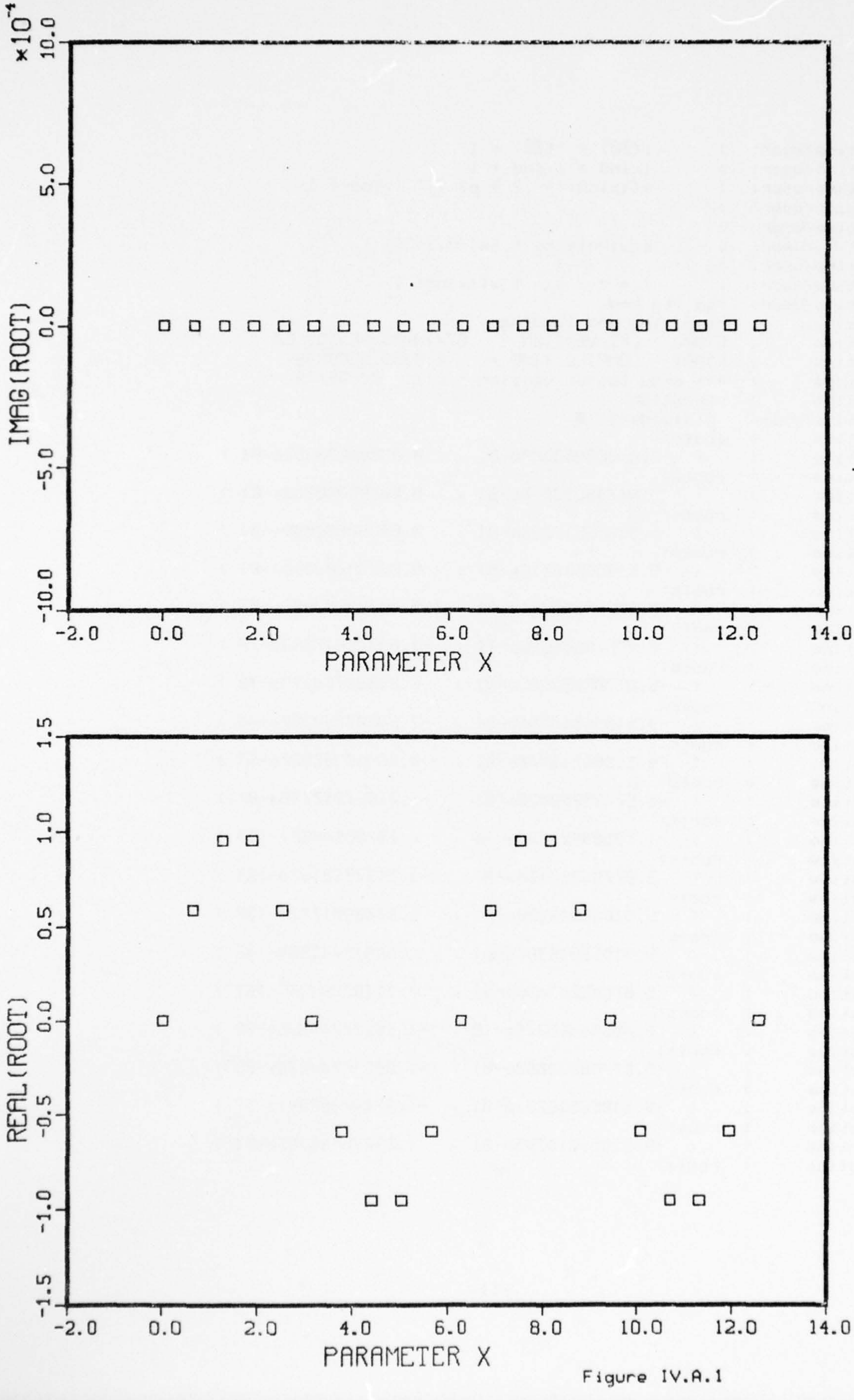

IMAG(ROOT)
*10^-4
10.0
5.0
0.0
-5.0
-10.0
-2.0
0.0
2.0
4.0
6.0
8.0
10.0
12.0
14.0
PARAMETER X
REAL(ROOT)
1.5
1.0
0.5
0.0
-0.5
-1.0
-1.5
-2.0
0.0
2.0
4.0
6.0
8.0
10.0
12.0
14.0
PARAMETER X

Figure IV.A.1

```
routine/user:  :       1(30) = 1(30) - 1
routine/user:  :       ixind = ixind + 1
routine/user:  :       x(ixind) = .2 * pi * ( ixind-1 )
routine/user:  :/
routine/user:  >f
routine/user:  :       equivalence ( ixind.1(29) )
routine/user:  :c
routine/user:  :       f = z - sin ( x(ixind) )
routine/user:  >go tty end
routine     :  compiling and loading
routine     :  FT004 - CFT VERSION -   07/14/79 SCHEDULER
routine     :  FT001 - COMPILE TIME =    0.0130 SECONDS
routine     :  *** cray loader version - c121  07/05/79
routine     :  executing
routine/user:  -plotmode=1  $
routine     :  roots:
routine     :    (  0.00000000000e-01 ,   0.00000000000e-01 )
routine     :  roots:
routine     :    (  5.87785252278e-01 ,   0.00000000000e-01 )
routine     :  roots:
routine     :    (  9.51056516284e-01 ,   0.00000000000e-01 )
routine     :  roots:
routine     :    (  9.51056516312e-01 ,   0.00000000000e-01 )
routine     :  roots:
routine     :    (  5.87785252351e-01 ,   0.00000000000e-01 )
routine     :  roots:
routine     :    (  8.97949606966e-11 ,  -1.89326617253e-29 )
routine     :  roots:
routine     :    ( -5.87785252205e-01 ,   6.31088724177e-30 )
routine     :  roots:
routine     :    ( -9.51056516256e-01 ,  -2.24207754292e-44 )
routine     :  roots:
routine     :    ( -9.51056516340e-01 ,  -4.08416990527e-53 )
routine     :  roots:
routine     :    ( -5.87785252423e-01 ,  -4.21687917729e-81 )
routine     :  roots:
routine     :    ( -1.79589921393e-10 ,   4.25795984001-109 )
routine     :  roots:
routine     :    (  5.87785252133e-01 ,  -1.51273121674-123 )
routine     :  roots:
routine     :    (  9.51056516229e-01 ,   1.07486017721-137 )
routine     :  roots:
routine     :    (  9.51056516367e-01 ,   1.58613441594-146 )
routine     :  roots:
routine     :    (  5.87785252496e-01 ,   7.11282799835-161 )
routine     :  roots:
routine     :    (  2.69356460379e-10 ,  -1.26217744835e-29 )
routine     :  roots:
routine     :    ( -5.87785252060e-01 ,  -1.26217744835e-29 )
routine     :  roots:
routine     :    ( -9.51056516201e-01 ,  -1.27447352891e-57 )
routine     :  roots:
routine     :    ( -9.51056516395e-01 ,   1.35835186182e-71 )
routine     :  roots:
```

```
routine      :    ( -5.87785252569e-01 ,   0.00000000000e-01 )
routine      :  roots:
routine      :    ( -3.59179842786e-10 ,   0.00000000000e-01 )
routine      :   all done
```

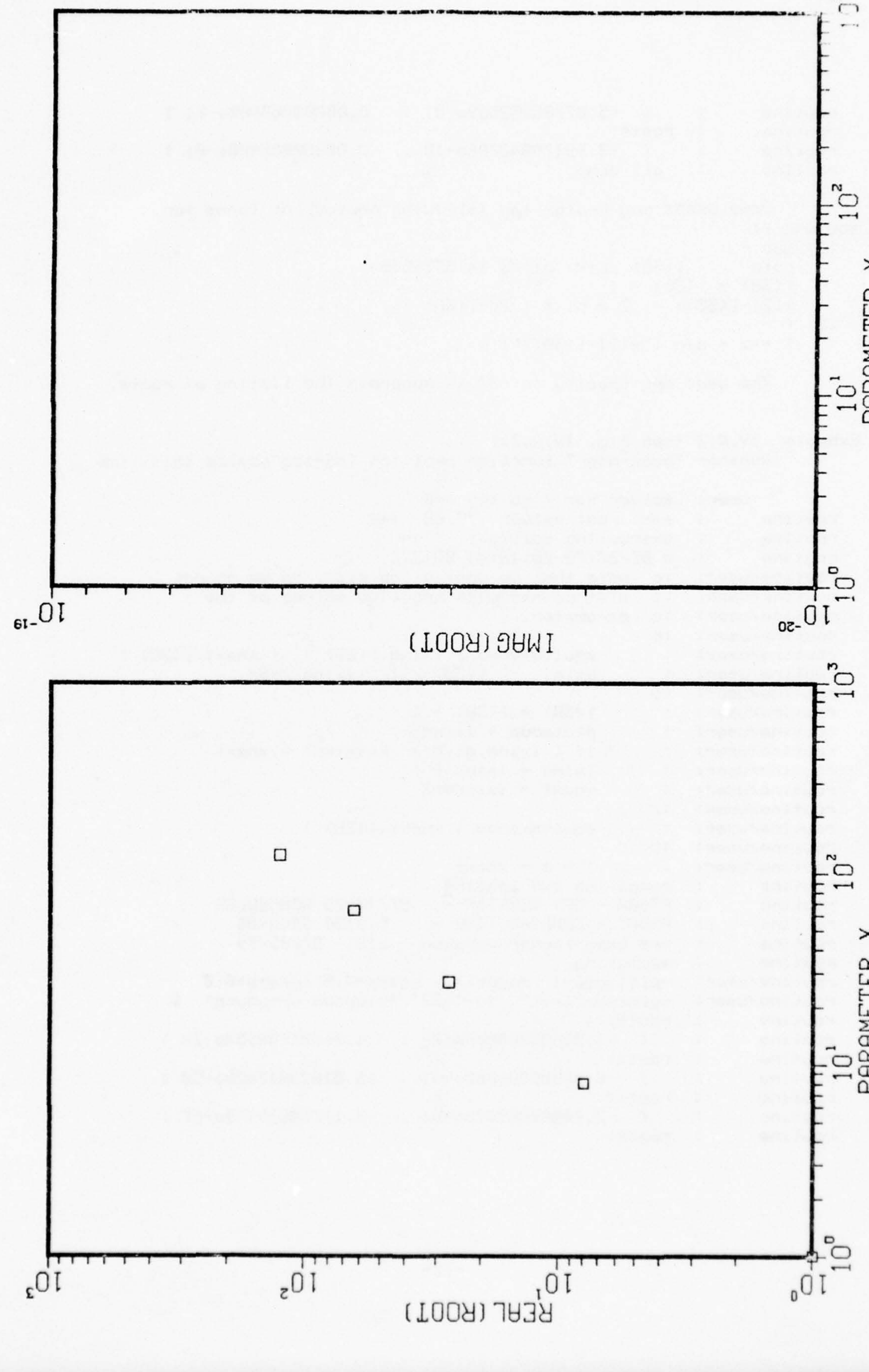
Some users may prefer the following equivalent forms for sub and f:

```
(1) sub -
    data      l(30) /21/, pi /3.1415926535/
    l(30) = l(30) - 1
    x(21-l(30)) = .2 * pi * ( 20-l(30) )
(2) f -
    f = z - sin ( x(21-l(30)) )
```

The user may specify "of=3" to suppress the listing of roots.

Example IV.A.2 (see Fig. IV.A.2):
Another "back-door" function plot, on log-log scales this time

```
        user:  solver sub f go tty end
routine      :  *** root solver  79.08  ***
routine      :  extracting solv/b/1
routine      :  c 07/22/79 20:19:41 001222
routine/user:  :c  note that we want a log plot, so we do not
routine/user:  :c  plot points with negative values of the
routine/user:  :c  parameter.
routine/user:  :c
routine/user:  :       equivalence ( ixind,l(29) ), ( xnext,l(28) )
routine/user:  :       data      l(30) /11/, ixind /-6/
routine/user:  :c
routine/user:  :       l(30) = l(30) - 1
routine/user:  :       plotmode = ixind
routine/user:  :       if ( ixind.gt.0 )  x(ixind) = xnext
routine/user:  :       ixind = ixind + 1
routine/user:  :       xnext = ixind**3
routine/user:  :/
routine/user:  :       equivalence ( xnext,l(28) )
routine/user:  :c
routine/user:  :       f = z - xnext
routine      :  compiling and loading
routine      :  FT004 - CFT VERSION -   07/14/79 SCHEDULER
routine      :  FT001 - COMPILE TIME =    0.0153 SECONDS
routine      :  *** cray loader version - c121  07/05/79
routine      :  executing
routine/user:  -plotmode=1  mapset=3  xsize=4.5  ysize=6.0
routine/user:  -plotid="test"  id="b22" "stephen au-yeung"  $
routine      :  roots:
routine      :    ( -1.25000000000e+02 ,   1.37001788954e-24 )
routine      :  roots:
routine      :    ( -6.40000000000e+01 ,  -8.01829413028e-50 )
routine      :  roots:
routine      :    ( -2.70000000000e+01 ,  -3.11734638189e-67 )
routine      :  roots:
```



IMAG(ROOT)
10^{-19}
10^{-20}
10^{0}
10^{1}
10^{2}
10^{3}
PARAMETER X
REAL(ROOT)
10^{0}
10^{1}
10^{2}
10^{3}
10^{0}
10^{1}
10^{2}
10^{3}
PARAMETER X

Figure IV.A.2

```
routine      :   ( -8.00000000000e+00 ,   7.67045853953e-93 )
routine      :  roots:
routine      :   ( -1.00000000000e+00 ,   6.16896886431-110 )
routine      :  roots:
routine      :   (  3.02922587605e-28 ,  -1.61229026582-137 )
routine      :  roots:
routine      :   (  1.00000000000e+00 ,   2.17066284129-165 )
routine      :  roots:
routine      :   (  8.00000000000e+00 ,  -1.75344747921-191 )
routine      :  roots:
routine      :   (  2.70000000000e+01 ,  -3.39941662484-218 )
routine      :  roots:
routine      :   (  6.40000000000e+01 ,   5.63600517340-231 )
routine      :  roots:
routine      :   (  1.25000000000e+02 ,  -3.99599503859-256 )
routine      :   all done
        user:  files f
routine      :    1375 rwe f105ro0x
routine      :   all done
        user:  netplot usc f105ro0x 1. box b22 steve
routine      :  destination:    ucb
routine      :          4 frames of output processed
routine      :  f105ro0x : file id is: steve - rxucb/rz
routine      :   all done
```

Example IV.A.3 (see Fig. IV.A.3), by Y.J. Chen:

In a 1d particle-hybrid simulation of the lower-hybrid drift instability (LHDI) in a uniform magnetic field, the quiet start Maxwellian loader was used to set up the warm, unmagnetized ion particles. Dispersion relations for the LHDI and the multi-beam instabilities were obtained from SOLVER. Initial guesses for roots were made in order to determine the approximate frequencies at which the desired roots should be obtained.

```
        user:  solver alib lhdmbi(LF)f disper sub mbilhd go imbilhd
routine      :  *** root solver  79.08  ***
routine      :  extracting solv/b/1
routine      :  c 07/22/79 20:19:41 001222
routine      :  compiling and loading
routine      :  FT004 - CFT VERSION -   07/14/79 SCHEDULER
routine      :  FT001 - COMPILE TIME =    0.0129 SECONDS
routine      :  *** cray loader version - c121  07/05/79
routine      :  executing
routine      :  routba   is the output file
routine/user:  >netplot f105ro0x 1. box b11 yjchen
routine      :  destination:    ucb
routine      :          6 frames of output processed
routine      :  f105ro0x : file id is: yjchen - rxucb/1a
routine/user:  >end
routine      :   all done
```

The contents of the various files are:

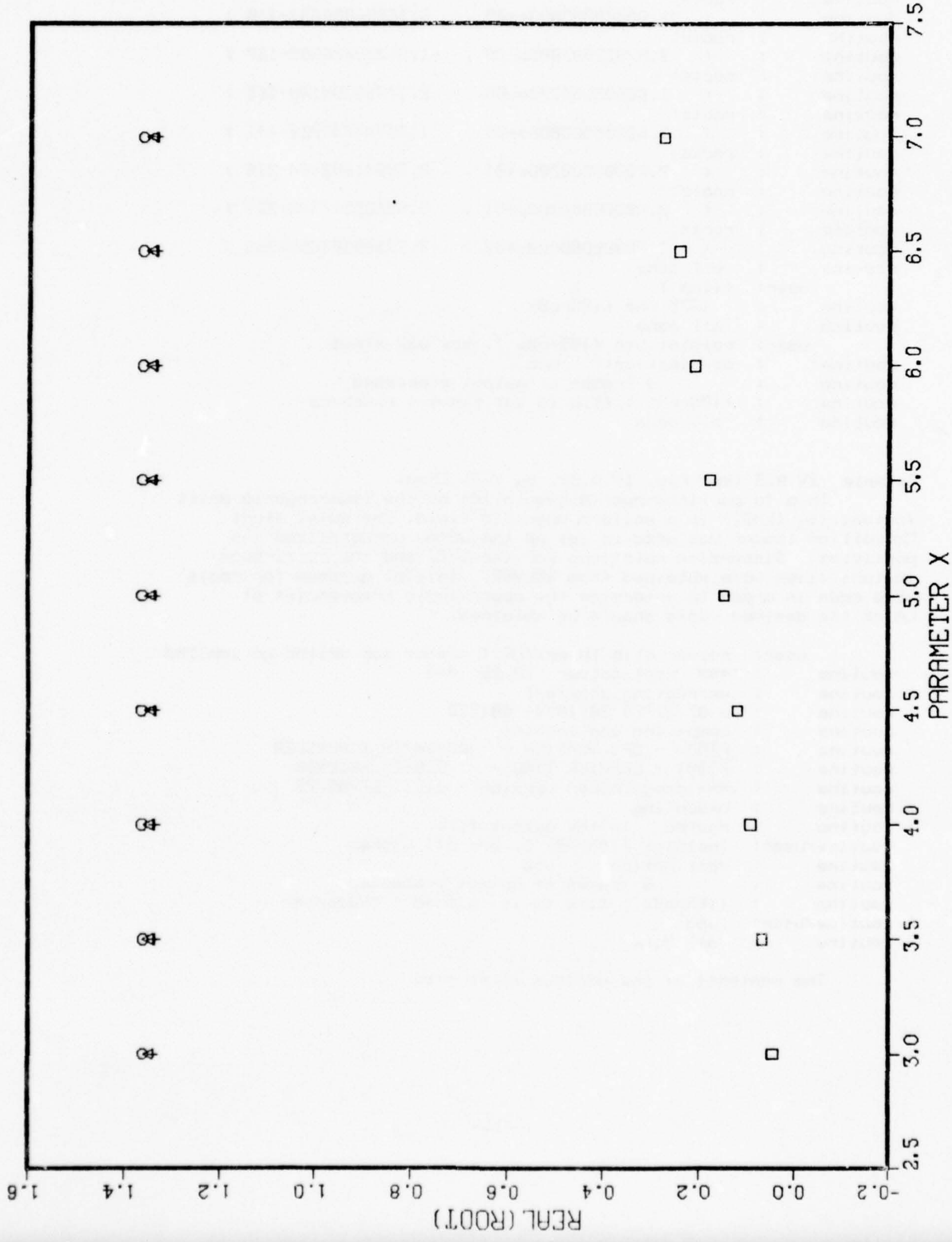


Figure IV.A.3 (a)

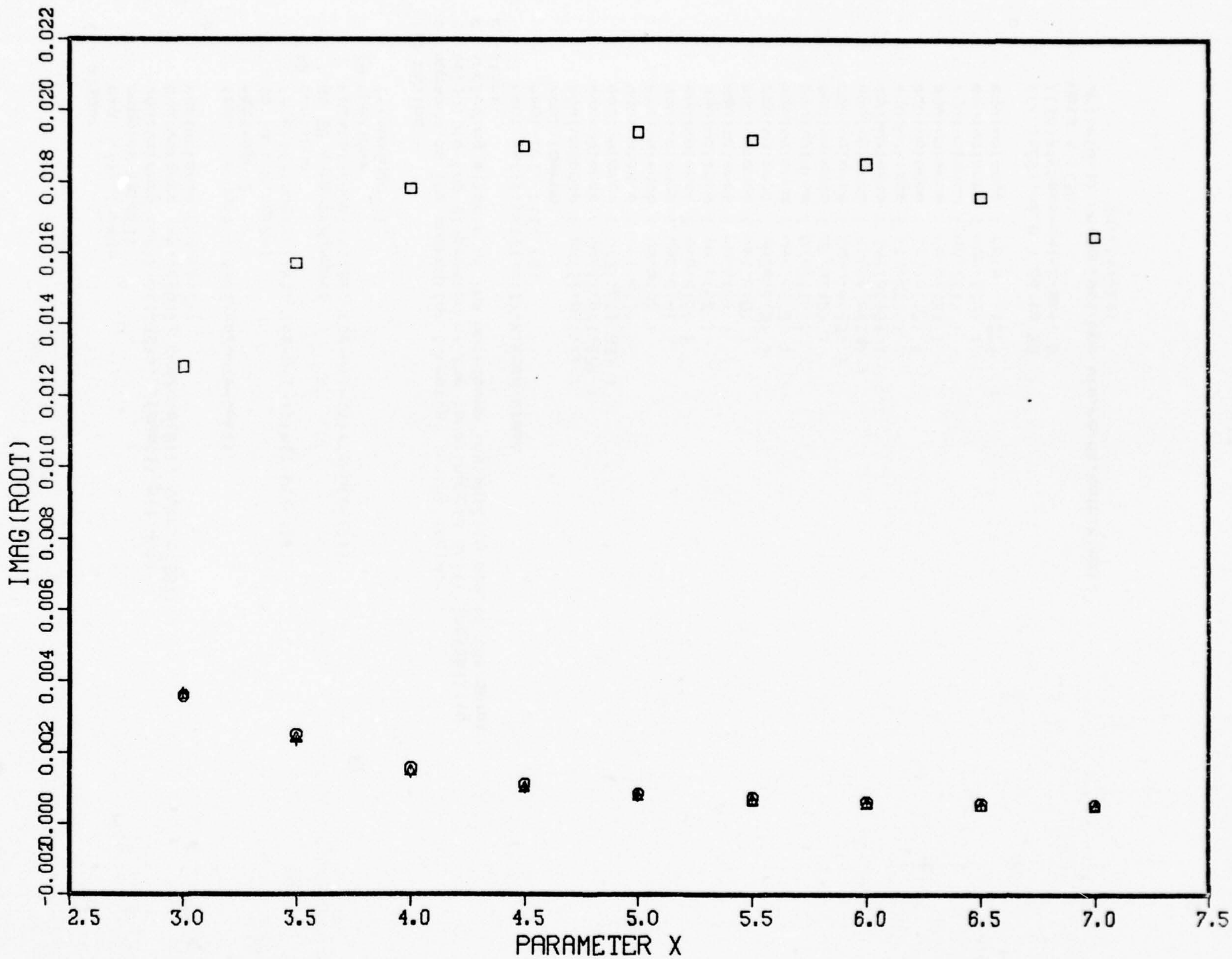


Figure IV.A.3 (b)

```
(1) disper -
      real v(1),lambda
      complex fsum(1)
      equivalence (v(1),p(1,33)), (fsum(1),p(1,49))
      equivalence (xel,c(8)), (xe2w,c(9)), (vek,c(10))
      equivalence (ngr,l(2))
c
      fsum(1)=-ngr*(1.+xel+xe2w/(z-vek))
      ngr2=ngr/2
      do 10 i=2,ngr2+1
      fsum(i)=fsum(i-1)+1./((z-v(i-1))*(z-v(i-1)))
   10 continue
      do 20 i=ngr2+2,ngr+1
      fsum(i)=fsum(i-1)+1./((z-v(i-1))*(z-v(i-1)))
   20 continue
      f=fsum(ngr+1)

(2) mbilhd -
c  appears as the subroutine "vvsetup" in the "solver".
c  calculate the dispersion of the lower hybrid drift instability
c  including effects of the multibeam instability due to the quiet
c  start.
      real xx(1),vx(1),v(1),K,Kmin,Klast
      real lni, lti, lsi
      real lambda
      equivalence ( xx(1),p(1,1) )
      equivalence ( vx(1),p(1,17) )
      equivalence ( v(1),p(1,33) )
      equivalence ( vti,c(1) )
      equivalence ( vd,c(3) )
      equivalence ( wpc,c(4) )
      equivalence ( vte,c(5) )
      equivalence ( ve,c(6) )
      equivalence ( rm,c(7) )
      equivalence ( xel,c(8) )
      equivalence ( xe2w,c(9) )
      equivalence ( vek,c(10) )
      equivalence ( b,c(11) )
      equivalence ( dk,c(12) )
      equivalence ( Kmin,c(13) )
      equivalence ( Klast,c(14) )
      equivalence ( lni,c(15) )
      equivalence ( lti,c(16) )
      equivalence ( lsi,c(17) )
      equivalence ( re,c(18) )
      equivalence ( nn,l(1) )
      equivalence ( ngr,l(2) )
      equivalence ( npts ,l(27)    )
c
      if( l(29).ne.0 ) go to 28
      l(30)=(Klast-kmin)/dk+1.5
      npts = l(30)
      write(6,1) nn,ngr,vti,wpc,vte,rm,dk,Kmin,Klast,
     .           lni,lti,lsi
```

```
    1 format(//,"ion:",//,"  nn=",i15,"    ngr=",i15,"    vti=",e14.5,/,
     .          "electron:",//,
     .          "  wpc=",e14.5,"   vte=",e15.5,/,
     .          "  mass ratio, rm=",e22.5,///,
     .          "  dk=",e15.5,"   kmin=",e14.5,"   klast=",e12.5,/,
     .          "  lni=",e14.5,"   lti=",e15.5,"   lsi=",e15.5,///)
c
c  set up the multibeam
       vmax=5.*vti
       dv=vmax/(nn-1)
       xx(1)=0.
      do 10 i=2,nn
      vv=(i-1.5)*dv
      fv=exp(-.5*(vv/vti)**2)
   10 xx(i)=xx(i-1)+fv
c
      df=xx(nn)/ngr
      j=1
      i1=ngr/2+1
      i2=i1-1
      do 12 i=1,ngr,2
      fv=i*df
   13 if( fv.lt.xx(j+1) ) go to 14
      j=j+1
      if( j.gt.nn-1 ) go to 25
      go to 13
   14 vv=dv*( j-1+( fv-xx(j) )/( xx(j+1)-xx(j) ) )
      vx(i1)=vv
      vx(i2)=-vv
      i1=i1+1
   12 i2=i2-1
   25 continue
c
      ngr2=ngr/2
c change vte to the electron larmor radius
   27 continue
      wcei=sqrt(wpc/rm)
      wcii=wcei*rm
      re=vte*wcei
      ve=vti*vti*(lni+lti)*wcii
      write(6,100)wcii,wcei,re,ve
  100 format("   wcii=",e13.5,"   wcei=",e13.5,"   re=",e16.5,/,
     .       "   ve=",e15.5,//)
   28 continue
      l(29) = l(29) + 1
      l(30) = l(30) - 1
      if ( l(30).lt.0 )  go to 36
c
c calculate coefficients of the dispersion relation
      k=klast-l(30)*dk
      write(6,29) k
   29 format(//,"k=",e12.5)
      x(l(29)) = k
      do 30 i=1,ngr
```

```
      v(i)=vx(i)*k
   30 continue
      vek=ve*k
      bk=vte*k
      b=bk*bk
      mm=n
c
      if( b.eq.0. ) go to 32
      bi0=bessei(0,b)
      bi1=bessei(1,b)
      lambda=bi0*exp(-b)
      a=( 1.-lambda )/b
      fle=b*( 1.-bi1/bi0 )
      go to 33
   32 lambda=1.0
      a=1.0
      fle=0.0
   33 xe1=a*wpc
      rli= lni - lsi*(1.-fle) - lti*fle
      xe2w=wpc*lambda*rli/(k*wcei)
c  guess the roots for the first time only
      if ( l(29).gt.1 )  go to 36
c
      j=2
   35 continue
      if ( j.gt.min0(ngr2,mm) )  return
      j1=ngr+1-j-10
      y(j)=v(j1)+cmplx(0.,.001)
      j=j+1
      go to 35
c
   36 continue
      do 40 j=1,mm
      y(j)=cmplx(real(y(j)),abs(aimag(y(j))))
   40 continue
```

```
(3) imbilhd -
n=4 maxit=300 h=.00002 of=2
l(1)=16384 l(2)=16384
c(1)=.141421  c(3)=0.0 c(4)=1.0
c(5)=0.00 c(6)=0.0 c(7)=1600.0 c(12)=0.5
c(13)=3.0 c(14)=7.0
c(15)=.075 c(16)=0.0 c(17)=0.0
y(1)=(.002,.001)
plotmode=1  plotid="lhdi" id="box b11" "y j chen"
xsize=9. ysize=6.75
 $
l(1)=nn, l(2)=ngr for giving y(j) in input file
c(1)=vt, c(3)=vd, c(4)=wpc
c(5)=vte, c(6)=ve, c(7)=rm, c(12)=dk
c(13)=kmin, c(14)=klast
c(15)=lni, c(16)=lti, c(17)=lsi
```

B.   Plotting a Function vs. One Argument

SOLVER can also plot a function vs. x, the real array described in Section IV.A. To tell SOLVER that the user wants a plot of the function rather than the roots of the function, the user need only assign the value 4, 5, or 6 to plotmode from the terminal or from the input file. The destination of the output when plotmode equals 4, 5, or 6 is the same as that when plotmode equals 1, 2, or 3, respectively. All other variables but "of" retain the same meaning and usage. The only output is a plot in this case.

Also note that within the function f, z is equivalent to x(i) where i corresponds to the number of times the function has been called. The structure can be viewed as follows:

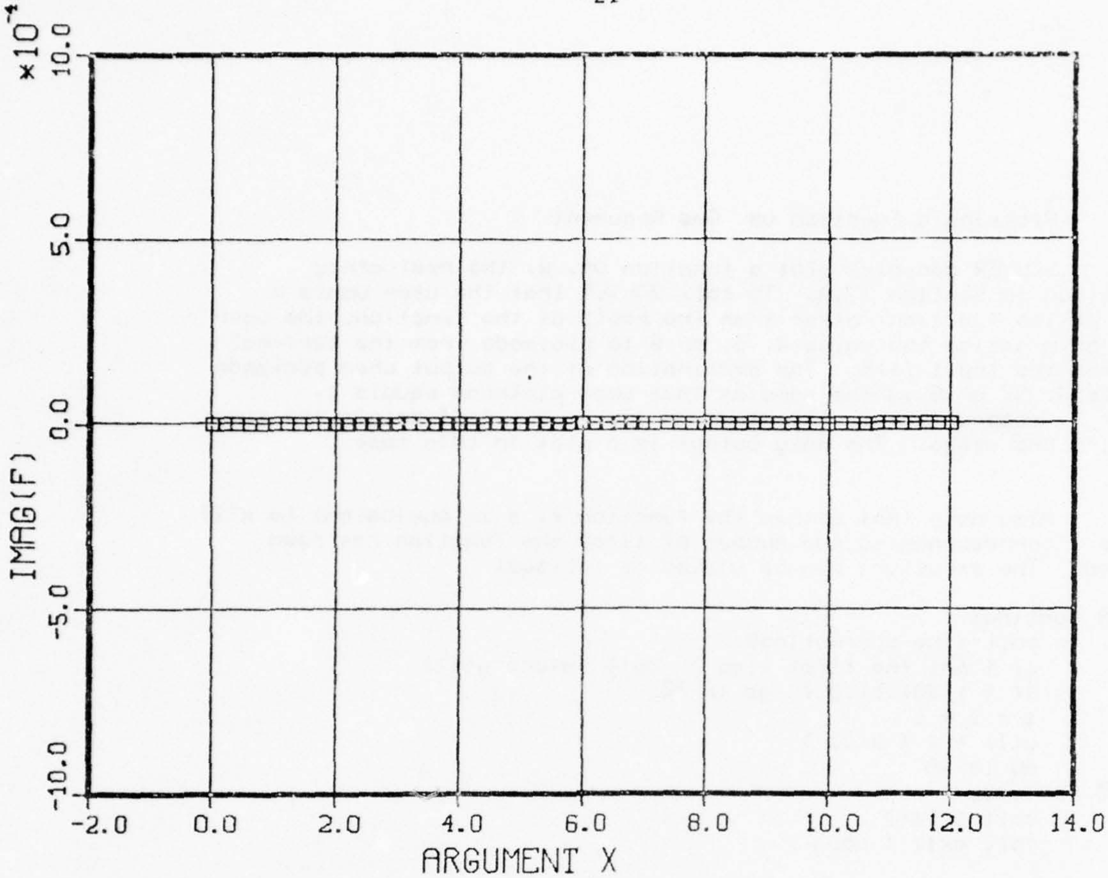
```
   40 continue
         call <the subroutine>
         if ( not the first time )  call <store y(1)>
         if ( l(30).lt.0 )  go to 70
         i = i + 1
         y(1) = f ( x(i) )
         go to 40
   70 continue
         call <plot>
         call exit ( dbug )
```

Example IV.B.1 (see Fig. IV.B.1):

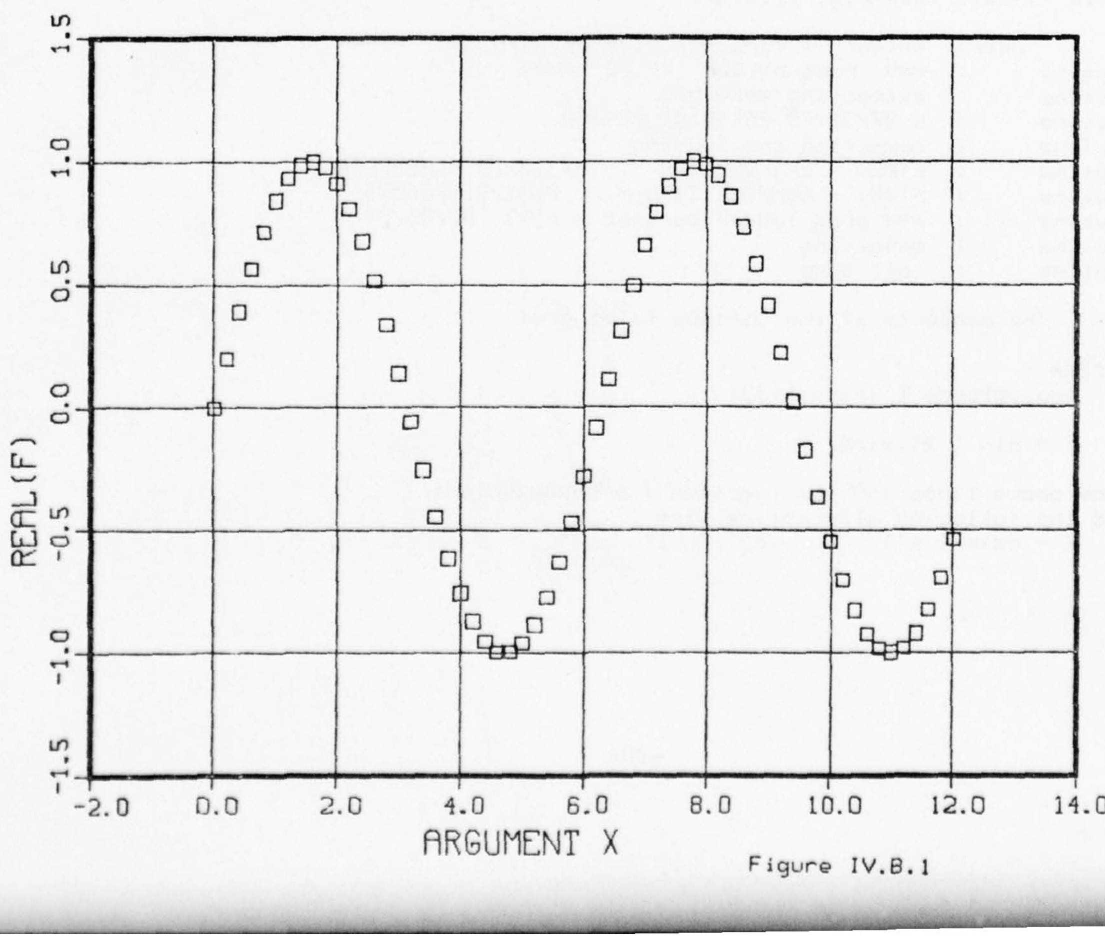
```
              user:  solver  f f1/s sub s1/s go g1/d end
  routine       :  *** root solver  79.08  ***
  routine       :  extracting solv/b/1
  routine       :  c 07/22/79 20:19:41 001222
  routine       :  compiling and loading
  routine       :  FT004 - CFT VERSION -   07/14/79 SCHEDULER
  routine       :  FT001 - COMPILE TIME =    0.0129 SECONDS
  routine       :  *** cray loader version - c121  07/05/79
  routine       :  executing
  routine       :   all done
```

The contents of the various files are:

```
(1) f1/s -
      equivalence ( ixind,l(29) )
c
      f = sin ( x(ixind) )
c
c  the above lines for the function f are equivalent
c  to the following alternative line
c     f = csin ( z )
```



×10⁻⁴
10.0
5.0
0.0
-5.0
-10.0
IMAG(F)
-2.0
0.0
2.0
4.0
6.0
8.0
10.0
12.0
14.0
ARGUMENT X



1.5
1.0
0.5
0.0
-0.5
-1.0
-1.5
REAL(F)
-2.0
0.0
2.0
4.0
6.0
8.0
10.0
12.0
14.0
ARGUMENT X

Figure IV.B.1

```
(2) s1/s -
      equivalence ( ixind,1(29) )
      data      1(30) /21/, ixind /0/
c
      1(30) = 1(30) - 1
      ixind = ixind + 1
      x(ixind) = .2 * (ixind-1 )

(3) g1/d -
1(30)=61
plotmode=4  grids=1
of=3
 $
```

C. Plotting Roots (a Function) vs. Two Parameters (Arguments)

A pair of contour plots will be produced in this case. The second parameter (argument) is to be stored in w, a real array of size 1024. The user should store values into x and w in the subroutine. In addition to x and w, two more variables have to be set at execution time; they are:

nx - Number of elements in x (defaults to 1024).

nw - Number of elements in w (defaults to 25).

If the product of nx and nw is greater than 1024x25=25600, nx and nw will be set back to their default values. It is important that w be the more rapidly varying parameter of the two (x is the slowly varying parameter). This point will become clear upon examination of the examples which follow.

Example IV.C.1 (see Fig. IV.C.1):
In this example, the fastest growing "tilting mode" root of a weak ion ring - plasma system is identified and contours of its real frequency and growth rate normalized to the ion cyclotron frequency are plotted as functions of two parameters. These are the ring strength "qr" and the background plasma density "zni". Logarithmic scales are used, and (in the stable region at the lower right) the frequency is set to zero when the growth rate is essentially zero, since there is then no "fastest growing mode". Note that the roots are reordered since the contour plotter can only plot one root, and we have not arranged to "follow" individual roots as parameters change. The constant "cay" depends upon the differencing scheme being used. (A. Friedman, unpublished)

```
        user:  solver f f6/s sub s6/s go g6/d
  routine   :  *** root solver 79.08 ***
  routine   :  extracting solv/b/l
```

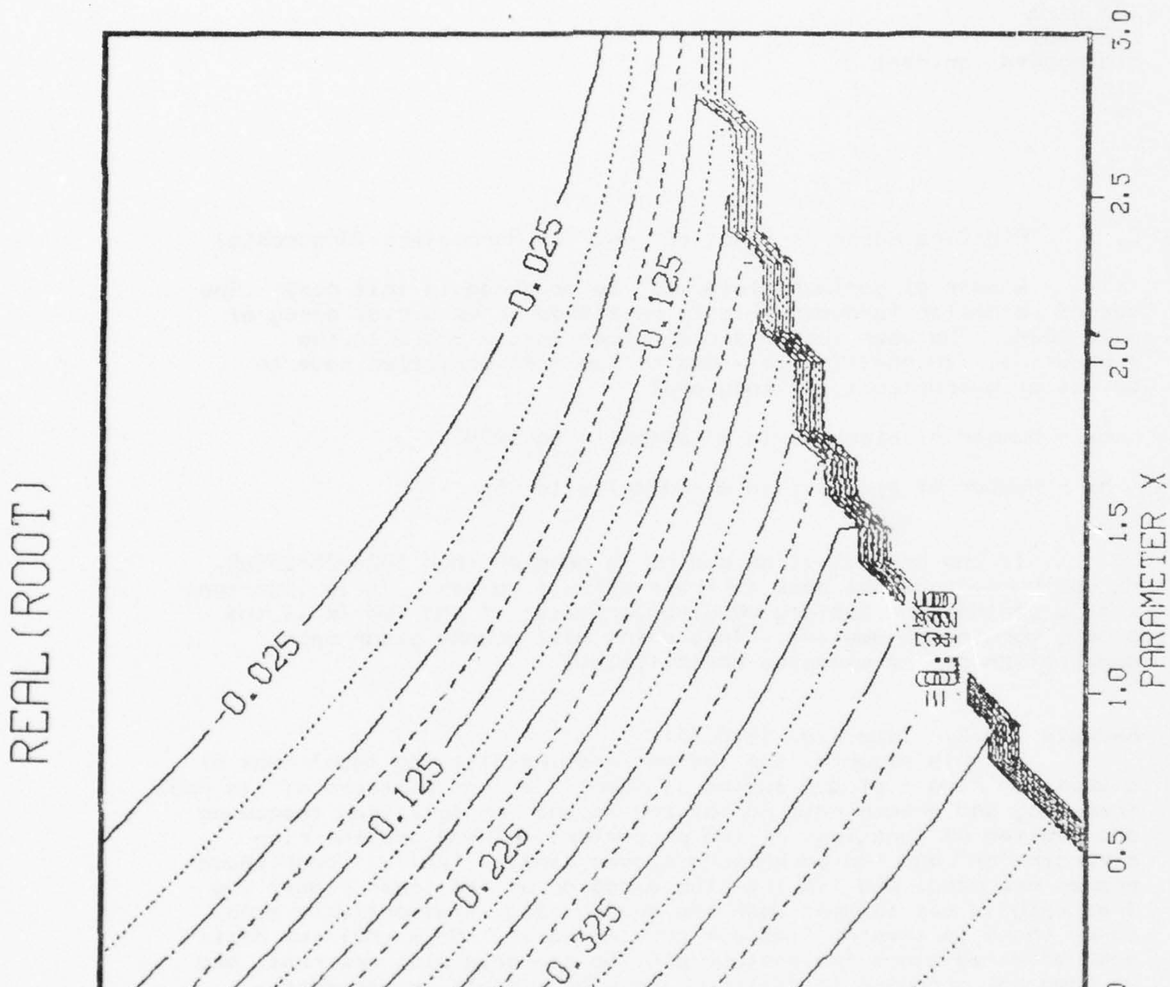


Figure IV.C.1 (a)

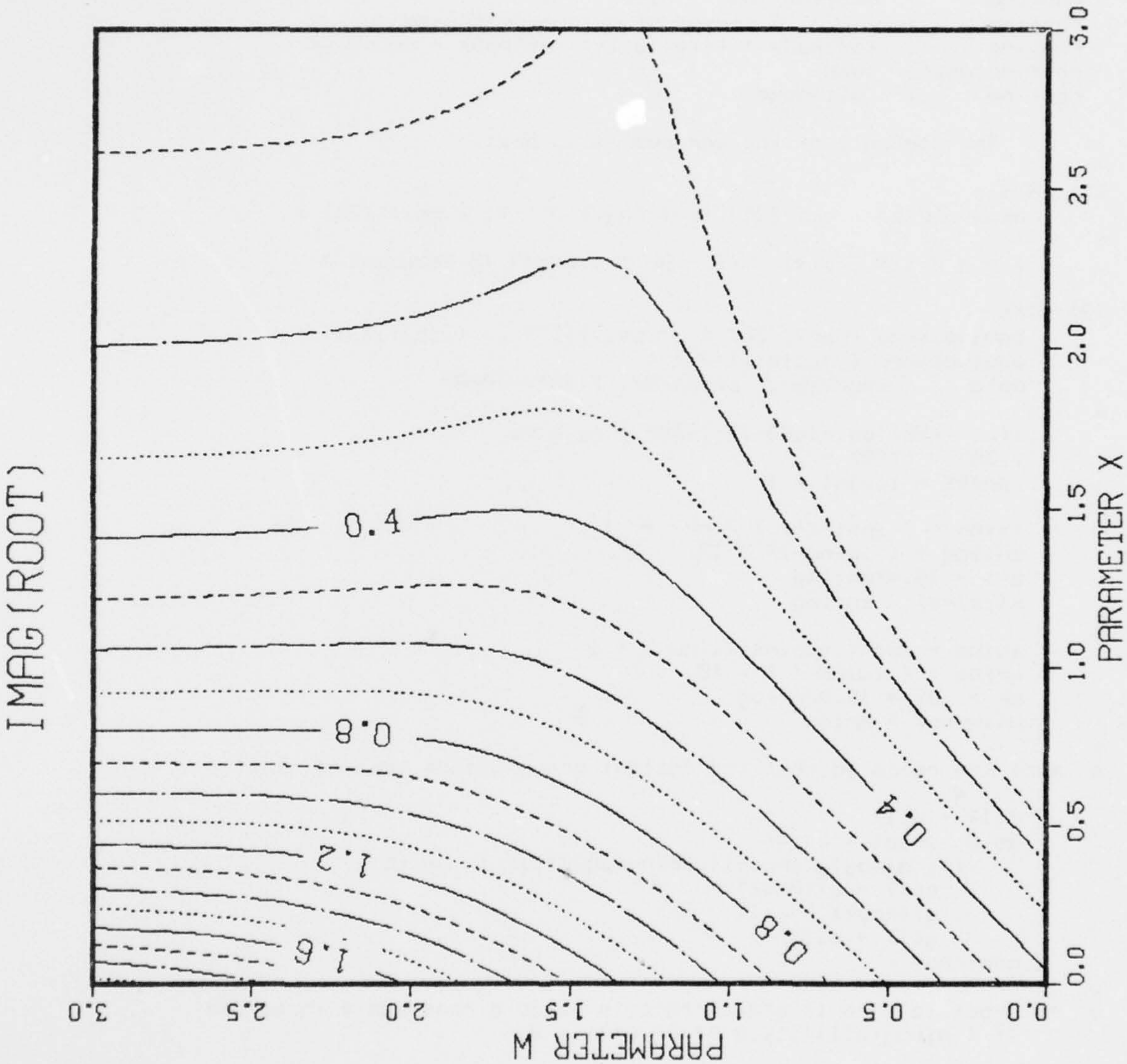


Figure IV.C.1 (b)

```
routine      :  c 07/28/79 11:21:15 001222
routine      :  compiling and loading
routine      :  FT004 - CFT VERSION -    07/14/79 SCHEDULER
routine      :  FT001 - COMPILE TIME =     0.0245 SECONDS
rotuine      :  *** cray loader version - c121  07/05/79
routine      :  executing
routine/user:  >netplot f105ro0x 1.
routine      :  box number:      b22
routine      :  destination:     ucb
routine      :            6 frames of output processed
routine      :  f105ro0x : file id is: f105ro0x - rxucb/4K
routine/user:  >end
routine      :   all done
```

The contents of the various files are:

```
(1) f6/s -
      equivalence ( qr,l(26) ), ( cay,l(27) ), ( zni,l(28) )
c
      f = z * z * ( 1.+z )**2 - qr * ( z*z/8.+2.*cay/zni )

(2) s6/s -
      equivalence ( qr,l(26) ), ( cay,l(27) ), ( zni,l(28) )
      equivalence ( ipoint,l(29) )
      data       cay/.25/, ipoint/0/, l(30)/-1000/
c
      if ( l(30).eq.-1000 )  l(30) = nx * nw
      l(30) = l(30) - 1
      ipoint = ipoint + 1
c
      ixind = ( ipoint-1 ) / nx  +  1
      znilog = ( ixind-1) / 10.
      zni = 10.**znilog
      x(ixind) = znilog
c
      iwind = mod ( ipoint-1, nw ) + 1
      qrlog = ( iwind-1 ) / 10.
      qr = .36 * 10.**qrlog
      w(iwind) = qrlog
c
c  sort the roots so that the fastest growing mode appears first
      n = 4
      c(1) = y(1)
      do 2  iroot = 2, 4
         if ( aimag(y(iroot)).le.aimag(c(1)) )  go to 2
           c(1) = y(iroot)
           y(iroot) = y(1)
           y(1) = c(1)
    2 continue
c
c  set root to zero if growth rate is below a reasonable threshold
      if ( aimag(y(1)).lt.0.01 )  y(1) = 0.

(3) g6/d -
nx=31  nw=31
plotmode=1  mapset=4
xsize=6. ysize=6.
of=3
 $
```

Example IV.C.2 (see Fig. IV.C.2):

```
          user:  solver f f5/s sub s5/s go g5/d
routine       :  *** root solver  79.08  ***
routine       :  extracting solv/b/1
routine       :  c 07/28/79 11:21:15 001222
routine       :  compiling and loading
routine       :  FT004 - CFT VERSION -   07/14/79 SCHEDULER
routine       :  FT001 - COMPILE TIME =    0.0245 SECONDS
rotuine       :  *** cray loader version - c121  07/05/79
routine       :  executing
routine/user:  >netplot f105ro0x 1.
routine       :  box number:     b22
routine       :  destination:    ucb
routine       :         6 frames of output processed
routine       :  f105ro0x ; file id is: f105ro0x - rxucb/\;
routine/user:  >end
routine       :   all done
```

The contents of the various files are:

(1) f5/s -

```
      equivalence ( ixind,l(29) ), ( iwind,l(28) )
c
      f = 0
      rsq = ( abs(x(ixind))-.5 )**2 + .25*w(iwind)**2
      if ( rsq.ge.0.25 )  return
      f = cmplx ( sqrt(.25-rsq), sqrt(rsq) )
      if ( x(ixind).gt.0. )  f = -f
```

(2) s5/s -

```
      equivalence ( ixind,l(29) ), ( iwind,l(28) )
      equivalence ( denx,l(27) ), ( denw,l(26) )
      data      l(30)/-1000/, ixind/0/, iwind/25600/
c
      if ( l(30).ne.-1000 )  go to 10
         l(30) = nx * nw
         denx = nx / 2
         denw = nw / 2
c
   10 continue
      l(30) = l(30) - 1
      if ( iwind.lt.nw )  go to 20
         ixind = ixind + 1
         x(ixind) = 1.2 * ( ixind-1-denx ) / denx
         iwind = 0
   20 continue
         iwind = iwind + 1
         w(iwind) = 1.2 * ( iwind-1-denw ) / denw
```

(3) g5/d -

```
plotmode=4
mapset=4
nx=81  nw=81
xsize=7. ysize=7.
of=3
 $
```

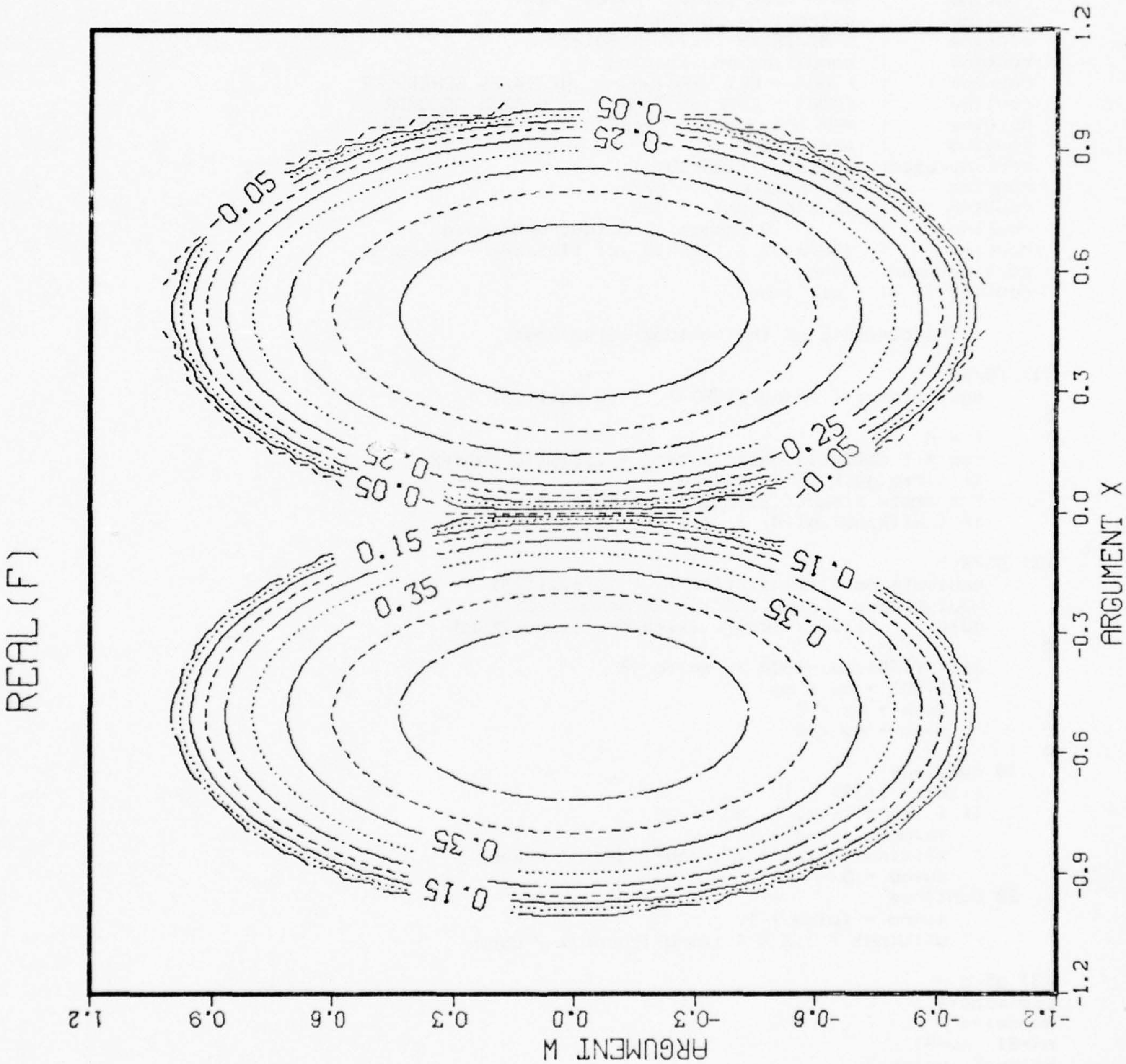


Figure IV.C.2 (a)

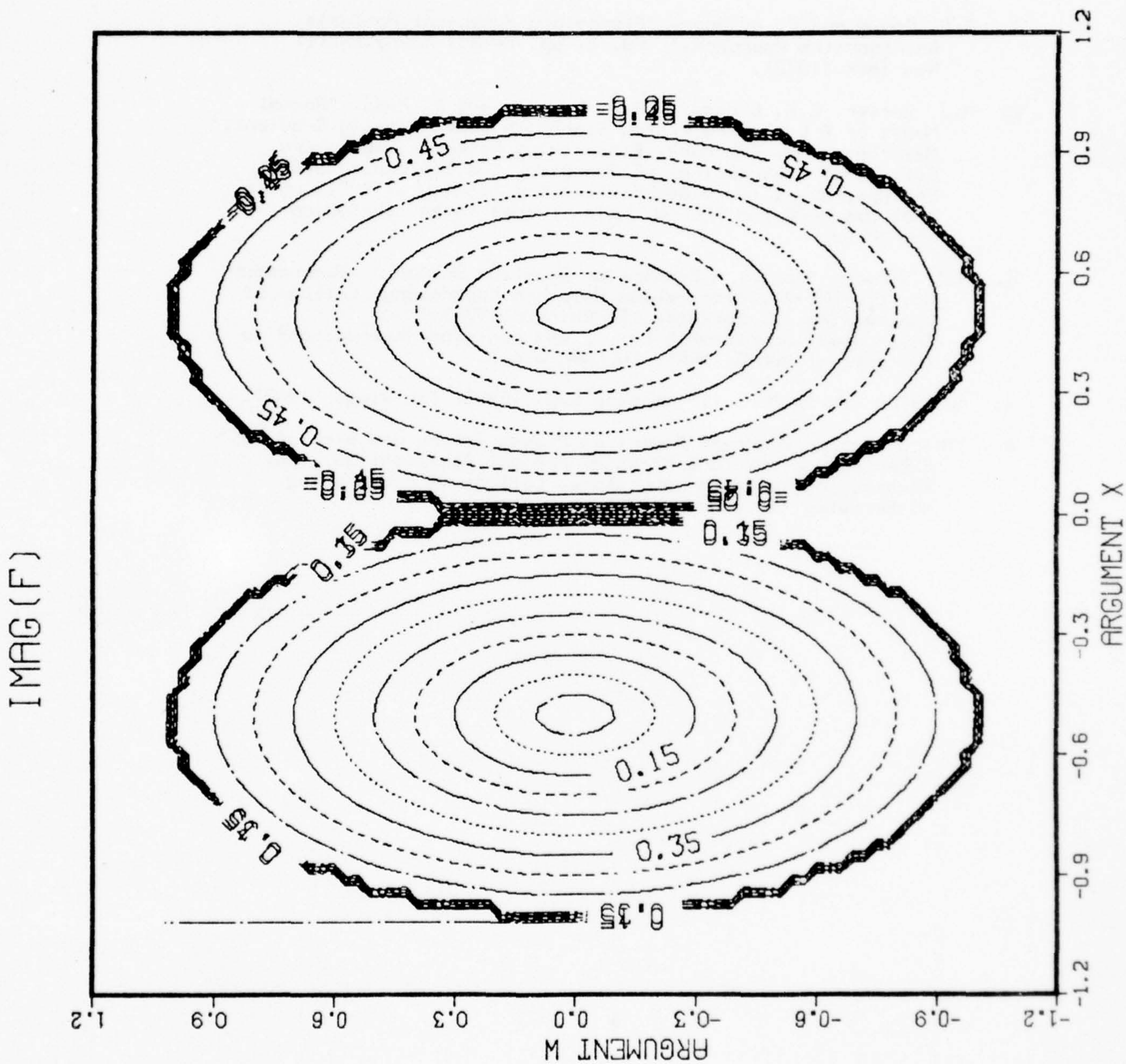


Figure IV.C.2 (b)

References

1. S.D. Conte and C. de Boor, "Elementary Numerical Analysis: An Algorithm Approach," Ed. 2, pp. 74-83, McGraw-Hill, New York (1972).

2. M.J. Gerver, C.K. Birdsall, A.B. Langdon and D. Fuss, "Normal Modes of A Loss Cone Plasma Slab with Steep Density Gradient," Memorandum No. ERL-M602, Electronics Research Laboratory, College of Engineering, UC Berkeley, (24 September 1976). See Appendix I for a description and listing of ROOTS which includes MRAF and FOLLOW, which traces roots, including branching.

3. M.J. Gerver, "Roots, A Dispersion Equation Solver," Memorandum No. ERL-M77-27, Electronics Research Laboratory, College of Engineering, UC Berkeley (31 October 1976). This report complements Ref. 2 and contains instructions to the user of ROOTS (MRAF, FOLLOW, etc.).

4. "DISSPLA User's Manual," Version 8.0, ISSCO, San Diego (1978).

5. "Third Quarter Progress Report on Plasma Theory and Simulation," ERDA Contract EY-76-S-03-0034, Project Agreement No. 128, Electronics Research Laboratory, College of Engineering, UC Berkeley (1 October 1977).

APPENDIX I. The Structure of The Library Solver

As mentioned in Section I, SOLVER is a library that contains all files needed by the SOLVER program.

Entering "solver / t v" from the terminal causes "solver/x", the first entry in SOLVER, to be executed. A dropfile named +solver<s>, where <s> is the current suffix (channel) that SOLVER is running under, is created. The next step is extracting the binary library "solv/b/l" from "solver" (if "solv/b/l" is not yet in the user's active file area). The prompt ">" will then appear and the user proceeds to enter commands.

The rest of the files in the SOLVER library are: the latest version of this documentation and the source files. Some of these files are part of the SOLVER controller and the rest are used in the generated program. A list of routines and corresponding source files follows:

(1) The SOLVER controller program -

file name	routine name
solver/s	main. (the main program)
cmds/s	iscmd
digit/s	digit
fexist/s	fexist
funct/s	getfct
inite/s	inite
letter/s	letter
prog/s	putprog
readf/s	readfct
remove/s	remove
savef/s	savef
scncmd/s	scancmd
sfmsge/s	shiftmsg
symove/s	symove

(2) The SOLVER generated program -

file name	routine name
driver/s	driver (the control routine of the generated program)
curves/s	curves
dest/s	destin
maptyp/s	maptype
root/s	muller
ncyc/s	ncyc
plot/s	rootplot
setf/s	setfile
step/s	stepsz
vvset/s	vvsetup

To modify SOLVER, the user need only recompile those source files that are modified and use BUILD to update "solv/b/l". If all the modified files are in item (2) above, the user need only update the SOLVER library "solv/b/l"; otherwise, the user must generate a new "solver/x" by using LDR as follows:

```
ldr i=solver/b,x=solver/x,lib=solv/b/l:fortlib
```

"solver/b" is the binary obtained by compiling main. (in file "solver/s") using CFT. If "solver/s" is not modified, "solver/b" can be generated by:

```
build ol. solv/b/l(LF)g. solver/b(LF)xg. main.(LF)end
```

APPENDIX I. The Structure of The Library Solver

As mentioned in Section I, SOLVER is a library that contains all files needed by the SOLVER program.

Entering "solver / t v" from the terminal causes "solver/x", the first entry in SOLVER, to be executed. A dropfile named +solver<s>, where <s> is the current suffix (channel) that SOLVER is running under, is created. The next step is extracting the binary library "solv/b/l" from "solver" (if "solv/b/l" is not yet in the user's active file area). The prompt ">" will then appear and the user proceeds to enter commands.

The rest of the files in the SOLVER library are: the latest version of this documentation and the source files. Some of these files are part of the SOLVER controller and the rest are used in the generated program. A list of routines and corresponding source files follows:

(1) The SOLVER controller program -

file name	routine name
solver/s	main. (the main program)
cmds/s	iscmd
digit/s	digit
fexist/s	fexist
funct/s	getfct
inite/s	inite
letter/s	letter
prog/s	putprog
readf/s	readfct
remove/s	remove
savef/s	savef
scncmd/s	scancmd
sfmsge/s	shiftmsg
symove/s	symove

(2) The SOLVER generated program -

file name	routine name
driver/s	driver (the control routine of the generated program)
curves/s	curves
dest/s	destin
maptyp/s	maptype
root/s	muller
ncyc/s	ncyc
plot/s	rootplot
setf/s	setfile
step/s	stepsz
vvset/s	vvsetup

To modify SOLVER, the user need only recompile those source files that are modified and use BUILD to update "solv/b/l". If all the modified files are in item (2) above, the user need only update the SOLVER library "solv/b/l"; otherwise, the user must generate a new "solver/x" by using LDR as follows:

```
ldr i=solver/b,x=solver/x,lib=solv/b/l:fortlib
```

"solver/b" is the binary obtained by compiling main. (in file "solver/s") using CFT. If "solver/s" is not modified, "solver/b" can be generated by:

```
build ol. solv/b/l(LF)g. solver/b(LF)xg. main.(LF)end
```

APPENDIX II. Listings of The Source Files

The following is a listing of all source programs used by SOLVER; files are current as of August 31, 1979, and are listed in the same order as in the table above:

```
c   main program of solver
c
       implicit   integer(a-z)
       common     /dabk/   keep, initprm
       data       keep /0/
       data       termin /0/
c
       integer    msg(10), symb(4), type(4), len
c
       integer    libuf(20), nlib, buf(22)
       data       buf /"  i=#b,x", "=#x,lib=", "solv/b/1", ":displa",
     .                  ":tv80lib", ":fortlib"/
       data       nlib /32/
       equivalence ( libuf(1), buf(3) )
c
c   initialization
       call dropfile ( 0 )
       call msgflag ( msgset, 2 )
       call msglink ( 59, 1 )
       call mprompt ( ">", 1 )
       call msgtor ( "*** root solver  79.08  ***", 32 )
c
c   extract binary modules from the library
       if ( fexist(libuf(1)).eq.0 )  go to 10
       call msgtor ( "extracting solv/b/1", 20 )
       call inite ( "lib", 60000000, 0., ires )
       call msgtoe ( "solver", 6, 1, 0, ires )
       call msgfre ( msg, len, 26, ires )
       call msgtor ( msg, len )
       call msgtoe ( "x solv/b/1", 10, 1, 0, ires )
       call msgfre ( msg, len, 32, ires )
       if ( len.gt.5 )  call msgtor ( msg, len )
       call bypass ( 1, 1 )
       call msgtoe ( "end", 3, 1, 0, ires )
       call msgfre ( msg, len, 16, ires )
c
    10 continue
          call msgtor ( " ", 1 )
c   scan commands
          call scancmd ( "#i", libuf, nlib, termin )
          go to  ( 20, 50, 40, 12, 40 ), termin+1
c   compile root solver
    12 continue
          termin = 4
    20 continue
          call msgtor ( "compiling and loading", 21 )
          call inite ( "cft", 6000000000, 0., ires )
          call destroy ( "#b" )
          call bypass ( 1, 1 )
          call msgtoe ( "i=#i,b=#b,l=0", 13, 0, 0, ires )
          call msgfre ( msg, len, 80, ires )
c   load root solver
    30 continue
          call inite ( "ldr", 6000000000, 0., ires )
```

```
          call destroy ( "#x" )
          call bypass ( 1, 1 )
          call msgtoe ( buf, nlib+16, 0, 0, ires )
          call msgfre ( msg, len, 80, ires )
c  execute root solver
          call msgtor ( "executing", 10 )
   40 continue
          call inite ( "#x", 6000000000, 0., ires )
          if ( ires.ne.0 )' go to 45
          call bypass ( 1, 1 )
          nc = -1
          if ( termin.eq.4 )  nc = 8
          call msgtoe ( initprm, nc, 0, 0, ires )
          call msgfre ( msg, len, 80, ires )
          call getsymb ( symb, type, nc, msg, len, 4 )
          if ( symb(3).ne."all" .or. symb(4).ne."done" )
     -       call msgtor ( msg, len )
          go to 10
   45     continue
             call msgtor ( "program not yet generated", 30 )
             go to 10
c
c  destroy temporary files
   50 continue
          call destroy ( "#i" )
          call destroy ( "#b" )
          call destroy ( "#x" )
          call destroy ( "1#b" )
          call destroy ( "$out" )
          call destroy ( "disout" )
          if ( keep.eq.0 )  call destroy ( "solv/b/1" )
          call exit
c
      end
```

```
      logical function  digit ( ch )
      integer   ch
c
c  digit returns true if ch is a digit
c
      integer   zero, nine
      data      zero / 60b /, nine / 71b /
c
      digit = .true.
      if ( ch.ge.zero .and. ch.le.nine )  return
      digit = .false.
      return
      end
```

```
        call destroy ( "#x" )
        call bypass ( 1, 1 )
        call msgtoe ( buf, nlib+16, 0, 0, ires )
        call msgfre ( msg, len, 80, ires )
c   execute root solver
        call msgtor ( "executing", 10 )
   40 continue
        call inite ( "#x", 6000000000, 0., ires )
        if ( ires.ne.0 ) go to 45
        call bypass ( 1, 1 )
        nc = -1
        if ( termin.eq.4 )  nc = 8
        call msgtoe ( initprm, nc, 0, 0, ires )
        call msgfre ( msg, len, 80, ires )
        call getsymb ( symb, type, nc, msg, len, 4 )
        if ( symb(3).ne."all" .or. symb(4).ne."done" )
     -     call msgtor ( msg, len )
        go to 10
   45   continue
           call msgtor ( "program not yet generated", 30 )
           go to 10
c
c   destroy temporary files
   50 continue
        call destroy ( "#i" )
        call destroy ( "#b" )
        call destroy ( "#x" )
        call destroy ( "1#b" )
        call destroy ( "$out" )
        call destroy ( "disout" )
        if ( keep.eq.0 )  call destroy ( "solv/b/1" )
        call exit
c
      end
```

```
      logical function  digit ( ch )
      integer   ch
c
c  digit returns true if ch is a digit
c
      integer   zero, nine
      data      zero / 60b /, nine / 71b /
c
      digit = .true.
      if ( ch.ge.zero .and. ch.le.nine )  return
      digit = .false.
      return
      end
```

```
      integer function  fexist  ( name )
      implicit  integer(a-z)
c
c  check if name is an existant file
c  fexist returns    0 => file exists
c                 else => file not exists
c
      call freeioc ( ioc )
      fexist = izopen ( ioc, name, len, acs )
      if ( fexist.eq.0 )  call izclose ( ioc, acs )
      return
      end
```

```
      subroutine  getfct ( buffer, nl )
      integer   buffer(9,1), nl
c
c gets the function or the subroutine and stores it in buffer.
c terminates when the first non-blank character is f followed by
c a blank or a equal sign, or is a slash
c
      logical   digit
      integer   line(80)
      integer   blank, equ, f, slash
      data      blank / 40b /
      data      equ / 75b /, f / 146b /
      data      slash / 57b /
c
      call mprompt ( ":", 1 )
      nl = 0
   20 continue
         nl = nl + 1
         call msgfrr ( buffer(1,nl), len, 72 )
         call zrjcharz ( line, buffer(1,nl), len )
         i = 0
   25    continue
            i = i + 1
            if ( line(i).eq.slash )  go to 40
            if ( line(i).eq.blank .or. digit(line(i)) )  go to 25
         if ( line(i).ne.f )  go to 20
         if ( line(i+1).ne.blank .and. line(i+1).ne.equ )  go to 20
c
   30 continue
      call mprompt ( ">", 1 )
      return
c
   40 continue
         nl = nl - 1
         go to 30
c
      end
```

```
      subroutine  inite ( name, time, prio, result )
      implicit  integer(a-z)
      integer   name, time, result
      real      prio
c
c initiates the controllee - name
c
      parameter ( nbeta=8 )
      integer   beta(nbeta)
      data      beta(1) /20000b/
      data      beta(6) /0/
      data      beta(7) /0/
      data      beta(8) /0/
c
      real      priority
      equivalence ( priority, beta(5) )
c
      beta(3) = name
      beta(4) = time
      priority = prio
      result = izcray ( beta, nbeta )
      return
      end
```

```
      logical function  letter ( ch )
      integer   ch
c
c  letter returns true if ch is a letter
c
      integer   a, z
      data      a / 141b /, z / 172b /
c
      letter = .true.
      if ( ch.ge.a .and. ch.le.z )  return
      letter = .false.
      return
      end
```

```
      subroutine  putprog ( ious, itext, buffer, nl, w )
      implicit  integer(a-z)
      integer   ious, itext, buffer(9,1), nl, w
c
c  writes the function f or the subroutine vvsetup onto the disk
c  file - itext
c
      real      h
      common    /miscbk/  n
      data      n /1/
c
      integer   fun(9), sub(9), fors(9,2)
      equivalence ( fun(1),fors(1,1) ), ( sub(1),fors(1,2) )
      data      fun /"       co","mplex fu","nction  ","f (z)",5*" "/
      data      sub /"       su","broutine","  vvsetu","p",5*" "/
c
      if ( w.gt.1 )  go to 10
      call freeus ( ious )
      call create ( ious, itext, 2, -1 )
c
c  write program on disk
      write(ious,101)
   10 continue
      write(ious,102)  (fors(i,w),i=1,9), n
      write(ious,103)  ((buffer(j,i), j=1,9), i=1,nl)
      write(ious,104)
      return
c
  101 format ( "      call driver" /
     .         "      end" )
  102 format ( 9a8 /
     .         "      complex   z" /
     .         "      integer    of, id(10), plotmode, plotid" /
     .         "      common    /miscbk/  n, maxit, h, ep1, ep2, of " /
     .         "      data      n /", i4, "/" /
     .         "      real      x(1024), w(1024)" /
     .         "      common    /plotbk/  id, mapset, plotmode," /
     .         "     .          plotid, x, w, xsize, ysize, inlen," /
     .         "     .          grids, nx, nw" /
     .         "      complex   c(30), p(1024,100), y(1024)" /
     .         "      integer   l(30)" /
     .         "      common    /parmbk/  l, c, p, y" )
  103 format ( 9a8 )
  104 format ( "      return" /
     .         "      end" )
c
      end
```

```
      subroutine  readfct ( ious, name, funct, nl, ierr )
      implicit  integer(a-z)
      integer   ious, name, funct(9,1), nl, ierr
      data      first /1/
c
c  reads function or subroutine from a file
c
      if ( first.eq.1 )  call freeus ( ious )
      first = 0
      ierr = 0
      call open ( ious, name, 0, ierr )
      if ( ierr.le.0 ) return
      nl = 0
   10 continue
         nl = nl + 1
         call rdline ( ious, funct(1,nl), len )
         if ( len.gt.-1 )  go to 10
         nl = nl - 1
         call close ( ious )
         return
c
      end
```

```
      subroutine  remove ( name, str, len )
      implicit  integer(a-z)
      integer   name, str(1), len
c
c  remove name from str.  len is the number of character in str.
c
      parameter ( cpw=8 )
      parameter ( blank=40b )
      integer   buf(25)
c
      call zmovechr ( buf, 0, str, 0, len )
c
c  compute the length of name
c
      n = zscanrne ( name, 0, cpw, blank ) + cpw
c
c  remove name
c
          loc = zskeybyt ( buf, 0, len, name, n )
          if ( loc.eq.0 )  go to 30
          if ( loc.eq.len )  go to 40
          call zmovechr ( str, 0, buf, 0, loc-1 )
          len = len - n - 1
          call zmovechr ( str, loc-1, buf, n+loc, len-loc+1 )
          return
c
   30 continue
          n = n + 1
          len = len - n
          call zmovechr ( str, 0, buf, n, len )
          return
c
c  not found
c
   40 continue
          write(59,101)  name
          return
c
  101 format ( "not found: ", a8 )
c
      end
```

```
      subroutine  savef ( name, funct, nl )
      implicit  integer(a-z)
      integer   name, funct(9,1), nl
c
c  save funct to the file "name"
c
      if ( fexist(name).ne.0 )  go to 5
         answer = "n"
         call mprompt ( "destroy existence file? ", 24 )
         call msgfrr ( answer, len, 1 )
         call mprompt ( ">", 1 )
         if ( answer.ne."y" )  return
    5 continue
      call freeus ( ious )
      call create ( ious, name, 2, -1, ierr )
      if ( ierr.ne.0 )  go to 20
      write(ious,101)  ((funct(j,i), j=1,9), i=1,nl)
      call close ( ious )
      return
c
   20 continue
         call msgtor ( "error creating file", 25 )
         return
c
  101 format ( 9a8 )
c
      end
```

```
      subroutine  scancmd  ( itext, libuf, nlib, termin )
      implicit  integer(a-z)
      integer   itext, libuf(1), nlib, termin
c
c this subroutine scans the input line and executes the
c given commands
c
      parameter ( ncmd=10 )
      integer   command(ncmd)
      data      command /
     .                   "sub",
     .                   "kbin",
     .                   "dlib",
     .                   "xeq",
     .                   "saves",
     .                   "savef",
     .                   "alib",
     .                   "end",
     .                   "go",
     .                   "f" /
      common    /miscbk/  n
c
      common    /dabk/    keep, initprm
c
      parameter ( maxsym=80 )
      integer   symb(maxsym), type(maxsym), buffer(10)
c
      integer   i, numsym
      data      i / 0 /, numsym / 0 /
c
      integer   funct(9,300), nl
      data      funct / "      f=", "0.", 7*" " /
      data      nl / 1 /
      integer   prog(9,300), npl
      data      prog / "c", 8*" " /
      data      npl / 1 /
c
      integer   buf(20)
c
      if ( i.lt.numsym )  go to 20
c
c main loop to interpret commands
c
   10 continue
        i = 0
        call msgfrr ( buffer, len, maxsym )
        call getnumb ( symb, type, numsym, buffer, len, maxsym )
        type(numsym+1) = 7
   20   continue
          i = i + 1
          go to  ( 100, 400, 20, 420, 420, 420, 420, 10 ), type(i)+1
c
  100   continue
          go to  ( 120, 300, 280, 260, 240, 230, 220, 200, 180,
```

```
     1               160, 140 ), iscmd(symb(i),command,ncmd)+1
c
c  controllee assumed
c
  120     continue
             call inite ( symb(i), 60000000000, 0., ires )
             if ( ires.ne.0 )  go to 130
             call bypass ( '1, 1 )
             ns = -1
             if ( type(i+1).eq.7 )  go to 125
                call shiftmsg ( buffer, len )
                ns = len
  125        continue
             call msgtoe ( buffer, ns, 0, 0, ires )
             call msgfre ( buffer, len, maxsym, ires )
             call getsymb ( symb, type, numsym, buffer, len, 4 )
             if ( symb(3).eq."all" .and. symb(4).eq."done" )  go to 10
             call msgtor ( buffer, len )
             go to 10
  130        continue
             call msgtor ( "invalid command disregarded", 30 )
             go to 20
c
c  f command
c
  140     continue
             if ( type(i+1).ne.0 )  go to  155
                call readfct ( ious, symb(i+1), funct, nl, ires )
                if ( ires.lt.0 )  go to 155
                i = i + 1
             go to 20
  155        continue
                call getfct ( funct, nl )
             go to 20
c
c  go command
c
  160     continue
             termin = 0
             call putprog ( iou, itext, funct, nl, 1 )
             if ( icall.eq.1 ) call putprog ( iou, itext, prog, npl, 2 )
             call close ( iou )
             icall = 0
             if ( type(i+1).ne.0 )  return
             if ( iscmd(symb(i+1),command,ncmd).gt.0 )  return
             i = i + 1
             initprm = symb(i)
             termin = 3
             return
c
c  end command
c
  180     continue
             termin = 1
```

```
            return
c
c  alib command
c
  200     continue
            call zmovechr ( buf, 0, libuf, 0, nlib )
            newlib = 0
  205       continue
              if ( type(i+1).gt.1 )  go to 210
              i = i + 1
              call symove ( libuf, newlib, symb(i) )
              if ( type(i+1).eq.1 )  go to 205
              call zmovechr ( libuf, newlib, ":", 0, 1 )
              newlib = newlib + 1
              go to 205
  210       continue
              call zmovechr ( libuf, newlib, buf, 0, nlib )
              nlib = nlib + newlib
            go to 20
c
c  savef  command
c
  220     continue
            if ( type(i+1).ne.0 )  go to 225
            i = i + 1
            call savef ( symb(i), funct, nl )
            go to 20
  225       continue
              call msgtor ( "file name required", 20 )
            go to 20
c
c  saves command
c
  230     continue
            if ( type(i+1).ne.0 )  go to 225
            i = i + 1
            call savef ( symb(i), prog, npl )
            go to 20
c
c  xeq command
c
  240     continue
            termin = 2
            if ( type(i+1).gt.0 )  return
            if ( iscmd(symb(i+1),command,ncmd).gt.0 )  return
            i = i + 1
            initprm = symb(i)
            termin = 4
            return
c
c  dlib command
c
  260     continue
            if ( type(i+1).ne.0 )  go to 20
```

```
               i = i + 1
               call remove ( symb(i), libuf, nlib )
               go to 260
c
c  kbin command
c
  280     continue
               keep = 1
               go to 20
c
c  sub command
c
  300     continue
               icall = 1
               if ( type(i+1).ne.0 )  go to 305
                  call readfct ( ious, symb(i+1), prog, npl, ires )
                  if ( ires.lt.0 )  go to 305
                  i = i + 1
                  go to 20
  305        continue
                  call getfct ( prog, npl )
               go to 20
c
c  symbol with more than 8 characters
c
  400     continue
               call msgtor ( "symbol too long", 15 )
               go to 20
c
c  set number of solutions
c
  420     continue
               n = symb(i)
               go to 20
c
        end
```

```
      subroutine  shiftmsg ( line, len )
      implicit  integer(a-z)
      integer   line(1), len
c
c shiftmsg shifts off the first symbol from line.  len is the
c number of characters in line.
c
      parameter ( blank=40b )
c
      sbk = zscanlne ( line, 0, len, blank )
      loc = zscanleq ( line, sbk, len-sbk, blank )
      nbk = zscanlne ( line, sbk+loc, len-sbk-loc, blank )
      len = len - sbk - loc - nbk
      call zmovechr ( line, 0, line, sbk+loc+nbk, len )
      return
c
      end
```

```
      subroutine  symove ( destin, loc, source )
      implicit  integer(a-z)
      integer   destin(1), loc, source
c
c  move characters from source to destin(loc) until a blank is
c  detected.  loc is the offset in characters of destin
c
      parameter ( blank=40b )
      parameter ( cpw=8 )
c
         n = zscanleq ( source, 0, cpw, blank )
         call zmovechr ( destin, loc, source, 0, n )
         loc = loc + n
         return
c
      end
```

```
      subroutine  driver
c
c  the driver of the solver generated program
c
      external    f
      complex     f, y(1024)
      integer     l(30), id(10), inlen, mapset, plotmode, plotid, of
      integer     grids, dbwg, first
      complex     c(30), p(1024,100)
      real        x(1024), w(1024)
      common      /parmbk/  l, c, p, y
      common      /plotbk/  id, mapset, plotmode, plotid, x, w,
     .                      xsize, ysize, inlen, grids, nx, nw
      common      /miscbk/  n, maxit, known, h, ep1, ep2, of
      namelist    /initbk/  n, maxit, known, h, ep1, ep2, of, l, c, p, y,
     .                      mapset, plotmode, plotid, x, w, xsize, ysize,
     .                      id, grids, dbug, nx, nw
      parameter ( npmax=1024, nrmax=25 )
      real        rootre(npmax,nrmax), rootai(npmax,nrmax)
c
c  default values
c
      data        maxit /100/
      data        known /0/
      data        h /.5/
      data        ep1 /0./
      data        ep2 /0./
      data        of /0/
      data        mapset /0/
      data        plotmode /0/
      data        plotid /0/
      data        xsize /6.0/
      data        ysize /4.5/
      data        id /"solver"/
      data        grids /0/
      data        dbug /0/
      data        nx /npmax/
      data        nw /nrmax/
c
      integer     line(10), symb(1), type(1)
      integer     namef(2)
      data        namef / 5r+root, 4rrout /
c
c  initializations
c
      call setfile ( namef, 2 )
      call dropfile ( namef(1) )
      call msgflag ( msg, 2 )
c
c  read input, if any
c
      if ( msg.eq.0 )  go to 15
         call msgfrr ( line, len, 80 )
         call getnumb ( symb, type, ns, line, len, 1 )
```

```
             if ( symb(1).ne."tty" )  go to 20
                call mprompt ( "-", 1 )
                read ( 59, initbk )
                call mprompt ( 0, 0 )
                go to 15
      20    continue
                call open ( 5, symb(1), 0, inlen )
                if ( inlen.lt..1 )  go to 35
                read ( 5, initbk )
c
c     create and write to output file, if of=1 or 2
c
      15 continue
         if ( of.lt.1 .or. of.gt.2 )  go to 35
         namef(2) = 400b*namef(2) + 140b
      10 continue
             namef(2) = namef(2) + 1
             call open ( 1, namef(2), 0, len )
             call close ( 1 )
             if ( len.gt.0 )  go to 10
         call create ( 6, namef(2), 2, -1 )
         write ( 59, 103 )  namef(2)
         write (  6, 104 )  n, maxit, known, h, ep1, ep2, (y(i), i=1,n)
c
c     check information
c
      35 continue
             n = min0 ( n, 1024 )
             if ( of.gt.2 .and. plotmode.eq.0 )  plotmode = 1
             iplot = plotmode
             if ( iplot.gt.3 .or. mapset.gt.3 )  n = 1
             np = nx
             nr = nw
             if ( np*nr.le.25600 .and. mapset.gt.3 )  go to 40
                np = npmax
                nr = nrmax
c
c     main loop
c
      40 continue
             call vvsetup
             if ( plotmode.gt.0 .and. first.eq.1 )
     .              call rootplot ( rootre, rootai, np, nr )
             if ( l(30).lt.0 )  go to 70
             first = 1
             if ( iplot.gt.3 )  go to 60
             num = n
             call muller ( known, num, y, maxit, h, ep1, ep2, f, kount )
             if ( of.gt.2 )  go to 40
             if ( of.eq.2 )  go to 50
             write ( 59, 101 )  ( y(i), i = 1, num+known )
             if ( kount.gt.maxit )  write ( 59, 102 )  kount
             if ( of.eq.0 )  go to 40
      50 continue
```

```
           write (  6, 101 )  ( y(i), i = 1, num+known )
           if ( kount.gt.maxit )  write (  6, 102 )  kount
           go to 40
c
c  evaluating function instead of finding roots
c
   60 continue
           ixind = ixind + 1
           y(1) = f ( x(ixind) )
           go to 40
c
c  terminating
c
   70 continue
           if ( iplot.le.0 )  call exit ( dbug )
           plotmode = iplot
           call rootplot ( rootre, rootai, np, nr )
           call donepl
           call exit ( dbug )
c
  101 format ( "roots:" / ( "   ( ",e19.11," , ",e19.11," )" ) )
  102 format ( "does not converge after ", i5, "  iterations" )
  103 format ( a8, " is the output file" )
  104 format ( "n      =", i4 / "maxit =", i4 / "known =", i4 /
     .         "h      =", f15.7 / "ep1    =", f15.7 / "ep2    =", f15.7 /
     .         "y      =", 2(2f15.7, ", ") / ( 7x, 2(2f15.7, ", ") ) )
      end
```

```
      subroutine  curves ( x, y, np, mark, nc )
      real        x(1), y(1024,1)
      integer     np, mark, nc
c
c  draw curves
c
      do 10  i = 1, nc
         call curve ( x(1), y(1,i), np, mark )
   10 continue
      return
      end
```

```
      subroutine  destin ( plotmode, plotid )
      integer   plotmode, plotid
c
c  plotmode = 1 => fr80
c           = 2 => prtplt
c           = 3 => tk
c  plotid for <fr80>    => id
c         for <prtplt> => line width
c         for <tk>     => baud rate
c
      mode = plotmode
      if ( mode.lt.1 )  return
      if ( mode.gt.3 )  mode = mod ( mode, 4 ) + 1
      go to  ( 10, 20, 30 ), mode
c
c  fr80 file
c
   10 continue
         call keep80 ( "roots", 3 )
         call fr80id ( plotid, 1, 1 )
         call plts
         return
c
c  printer
c
   20 continue
         call prtplt ( plotid )
         return
c
c  tektronix
c
   30 continue
         call tk ( 0, .1*plotid, 59 )
         return
c
      end
```

```
      subroutine  maptype ( mapset, npoint, xmin, xmax, ymin, ymax,
     .                      xsize, ysize )
      integer   mapset, npoint
      real      xmin, xmax, ymin, ymax, xsize, ysize
c
c  mapset = 0  => linear-linear
c         = 1  => linear-log
c         = 2  => log-linear
c         = 3  => log-log
c         = 4  => contour
c
      parameter ( nwsp=16384 )
      common    work(nwsp)
      data      scale /"scale"/
c
      map = mapset
      if ( map.lt.0 )  map = 0
      if ( map.gt.4 )  map = 0
      nxsize = ( xsize+1 ) / 2
      nysize = ( ysize+1 ) / 2
      xinc = stepsz ( xmin, xmax, 2.*nxsize, xd )
      yinc = stepsz ( ymin, ymax, 2.*nysize, yd )
      xcyc = xsize / ncyc ( xmin, xmax, xdec )
      ycyc = ysize / ncyc ( ymin, ymax, ydec )
      go to  ( 10, 20, 30, 40, 50 ),  map+1
c
   10 continue
         call graf ( xmin-xd, scale, xmax+xd, ymin-yd, scale, ymax+yd )
         return
c
   20 continue
         call ylog ( xmin-xd, xinc, ydec, ycyc )
         return
c
   30 continue
         call xlog ( xdec, xcyc, ymin-yd, yinc )
         return
c
   40 continue
         call loglog ( xdec, xcyc, ydec, ycyc )
         return
c
   50 continue
         xinc = xinc - 2. * xd / max0 ( 2*nxsize,4 )
         yinc = yinc - 2. * xd / max0 ( 2*nysize,4 )
         call graf ( xmin, xinc, xmax, ymin, yinc, ymax )
         call bcomon ( nwsp )
         call conlin ( 0, "solid", "labels", 1, 10 )
         call conlin ( 1, "dash", "nolabels", 1, 8 )
         call conlin ( 2, "chndot", "nolabels", 1, 5 )
         call conlin ( 3, "dot", "nolabels", 1, 3 )
         call conang ( 60. )
         return
c
      end
```

```
      subroutine  muller (Kn, n, rts, maxit, step, ep1, ep2, fn, Kount)
      complex  rts(1)
      complex  fn
c
c  this subroutine uses the muller's method to find the roots of
c  a function
c
c  parameters:
c    Kn      - number of roots previously computed, normally zero
c    n       - input:  number of roots desired
c              output: number of roots found
c    rts     - an array of initial estimates of all desired roots
c              set to zero if no better estimates are available
c    maxit   - maximum number of iterations per root allowed
c    step    - initial distance between x's
c    ep1     - relative error tolerance on x(i)
c    ep2     - error tolerance on f(x(i))
c    fn      - fn(z) is a external complex function which, for given
c              z, returns f(z)
c    Kount   - number of iterations of the last root.  if Kount
c              is equal to maxit, the routine will stop at once
c
c
      complex  rt, h, delfpr, frtdef, lambda, delf, dfprlm, num, den
      complex  g, sqr, frt, frtprv
c
c  initialization
      eps1 = amax1 (ep1, 1.e-12)
      eps2 = amax1 (ep2, 1.e-20)
      ibeg = Kn + 1
      iend = Kn + n
c
      do 100 i = ibeg, iend
         Kount = 0
c  compute first three estimates for root as
c  rts(i)+step, rts(i)-step, and rts(i)
  1      h = step
         rt = rts(i) + h
         index = 1
         go to 70
 10      delfpr = frtdef
         rt = rts(i) - h
         index = 2
         go to 70
 20      frtprv = frtdef
         delfpr = frtprv - delfpr
         rt = rts(i)
         index = 3
         go to 70
 30      index = 4
         lambda = 1.
c  compute next estimate for root
 40      delf = frtdef - frtprv
         dfprlm = delfpr * lambda
```

```
          num = -frtdef * (1. + lambda) * 2.
          g = (1. + lambda * 2.) * delf - lambda * dfprlm
          sqr = g * g + 2. * num * lambda * (delf - dfprlm)
          sqr = csqrt(sqr)
          den = g + sqr
          if (real(g) * real(sqr) + aimag(g) * aimag(sqr) .lt. 0.)
     1       den = g - sqr
          if (cabs(den) .lt. ep1)  den = 1.
          lambda = num / den
          frtprv = frtdef
          delfpr = delf
          h = h * lambda
          rt = rt + h
          if (Kount .gt. maxit)  go to 200
c
   70     Kount = Kount + 1
          frt = fn (rt)
          frtdef = frt
          do 71  j = 1, i-1
             den = rt - rts(j)
             if (cabs(den) .lt. eps2)  go to 79
             frtdef = frtdef / den
   71     continue
   75     go to (10, 20, 30, 80), index
   79     rts(i) = rt + .001
          go to 1
c
c  check for convergence
   80     if (cabs(h) .lt. eps1 * cabs(rt))  go to 100
          if (amax1(cabs(frt), cabs(frtdef)) .lt. eps2)  go to 100
c
c  check for divergence
          if (cabs(frtdef) .lt. 10. * cabs(frtprv))  go to 40
          h = .5 * h
          lambda = .5 * lambda
          rt = rt - h
          go to 70
  100 rts(i) = rt
      return
  200 continue
          n = i
          return
      end
```

```
      integer function  ncyc ( vmin, vmax, vdec )
      real         vmin, vmax, vdec
c
c  return the number of cycles needed for log scale
c
      parameter ( one=.999999999999 )
      parameter ( maxcyc=10 )
      parameter ( base=1.e-20 )
c
      delta = 0.
      if ( vmin.lt.1. )  delta = 1.
      min = alog10 ( amax1(vmin,base) ) - delta
      max = alog10 ( amax1(vmax,base) ) + one
      ncyc = max0 ( max-min, 1 )
      vdec = 10. ** min
      if ( vdec.gt.base*10. .or. ncyc.lt.maxcyc )  return
      ncyc = maxcyc
      vdec = 10. ** ( max-maxcyc )
      return
c
      end
```

```
      subroutine  rootplot ( rootre, rootai, npnew, nrnew )
      real     rootre(npnew,nrnew), rootai(npnew,nrnew)
c
c  plots rootre vs. x and/or w and rootai vs. x and/or w
c  rootre and rootai are the real and imaginary part of the
c  roots (or the value of the function. if so, stored in
c  rootre(1) and rootai(1)).  npnew is the maximum number of
c  points and nrnew is the maximum number of roots.
c
      integer   l(30), id(10), inlen, mapset, plotmode, plotid, of
      integer   grids
      real      x(1024), w(1024)
      complex   y(1024)
      complex   c(30), p(1024,100)
      common    /parmbk/  l, c, p, y
      common    /plotbk/  id, mapset, plotmode, plotid, x, w,
     .                    xsize, ysize, inlen, grids
      common    /miscbk/  n, maxit, known, h, ep1, ep2, of
c
      real      psize(2)
      parameter ( big=10.**100, small=-10.**100 )
      integer   first, npoint, line(11)
      data      remin, aimin, xmin, wmin /4*big/
      data      remax, aimax, xmax, wmax /4*small/
      data      line(11) /1h$/
      data      psize /11.,8.5/
      data      nx /0/, nw /25600/
c
      integer   labfre(2), labfai(2), labtre(2), labtai(2)
      integer   labwre(2), labwai(2), labx(2)
      integer   labpw(2), labax(2), labaw(2)
      data      labfre /"real(f)", " "/
      data      labfai /"imag(f)", " "/
      data      labpw  /"paramete", "r w"/
      data      labax  /"argument", " x"/
      data      labaw  /"argument", " w"/
      data      labx   /"paramete", "r x"/
      data      labwre /"real(roo", "t)"/
      data      labwai /"imag(roo", "t)"/
      data      labtre /" ", " "/
      data      labtai /" ", " "/
c
c  initializations
c
      if ( first.ne.0 )  go to 5
      first = 1
      nr = min0 ( n+known, nrnew )
      xsz = xsize
      ysz = ysize
c
c  check if page size should be 8.5"x11" or 11"x8.5"
c
      if ( xsz.eq.ysz )  go to 5
      if ( xsz.lt.ysz .and. ysz.le.psize(2)-1.5 )  go to 5
```

```
      if ( xsz.ge.ysz .and. xsz.gt.psize(2)-1.5 )  go to 5
         temp = psize(1)
         psize(1) = psize(2)
         psize(2) = temp
c
c  store the roots of the function value, y
c
   5 continue
      if ( l(30).lt.0 )  first = first + 1
      if ( first.gt.2 )  go to 20
      if ( mapset.gt.3 )  go to 12
      nx = nx + 1
      if ( nx.gt.npnew )  return
      do 10  i = 1, nr
         rootre(nx,i) = real ( y(i) )
         rootai(nx,i) = aimag ( y(i) )
         if ( remin.gt.rootre(nx,i) )  remin = rootre(nx,i)
         if ( remax.lt.rootre(nx,i) )  remax = rootre(nx,i)
         if ( aimin.gt.rootai(nx,i) )  aimin = rootai(nx,i)
         if ( aimax.lt.rootai(nx,i) )  aimax = rootai(nx,i)
  10 continue
      return
c
  12 continue
         if ( nw.lt.nrnew )  go to 14
            nx = nx + 1
            nw = 0
  14     continue
            nw = nw + 1
            rootre(nx,nw) = real ( y(1) )
            rootai(nx,nw) = aimag ( y(1) )
            if ( remin.gt.rootre(nx,nw) )  remin = rootre(nx,nw)
            if ( remax.lt.rootre(nx,nw) )  remax = rootre(nx,nw)
            if ( aimin.gt.rootai(nx,nw) )  aimin = rootai(nx,nw)
            if ( aimax.lt.rootai(nx,nw) )  aimax = rootai(nx,nw)
         return
c
c  find the minimum and maximum values of x and w
c
  20 continue
         npoint = min0 ( nx, npnew )
         do 30  j = 1, npoint
            if ( xmin.gt.x(j) )  xmin = x(j)
            if ( xmax.lt.x(j) )  xmax = x(j)
  30     continue
         if ( mapset.lt.4 )  go to 40
            do 35  j = 1, nw
               if ( wmin.gt.w(j) )  wmin = w(j)
               if ( wmax.lt.w(j) )  wmax = w(j)
  35        continue
            if ( remax-remin.lt.1.e-25 )  nore = 1
            if ( aimax-aimin.lt.1.e-25 )  noai = 1
            remin = wmin
            aimin = wmin
```

```
            remax = wmax
            aimax = wmax
c
c  figure out the title and name for the axes
c
   40 continue
         if ( plotmode.lt.4 .and. mapset.lt.4 )  go to 60
            if ( plotmode.lt.4 )  go to 50
               call move ( labx, labax, 2 )
               call move ( labwre, labfre, 2 )
               call move ( labwai, labfai, 2 )
               if ( mapset.lt.4 )  go to 60
               call move ( labwre, labaw, 2 )
               call move ( labwai, labaw, 2 )
               call move ( labtre, labfre, 2 )
               call move ( labtai, labfai, 2 )
               go to 60
   50       continue
               call move ( labtre, labwre, 2 )
               call move ( labtai, labwai, 2 )
               call move ( labwre, labpw, 2 )
               call move ( labwai, labpw, 2 )
c
c  getting ready to plot
c
   60 continue
         call setbox ( id, 80 )
         call gfsize ( 3, 2000000b )
         call destin ( plotmode, plotid )
         call nobrdr
c
c  plot the input file, if there is one
c
         if ( inlen.le.0 )  go to 90
            call page ( psize(1), psize(2) )
            call tablet ( "ltset", "long" )
            rewind 5
   70       continue
               read(5,1010)  ( line(i), i=1,10 )
               if ( ieof(5).ne.0 )  go to 80
               call ltline ( line )
               go to 70
   80    continue
            call endtab ( 0 )
            call dendpl ( 0 )
c
   90 continue
         xsz = amin1 ( xsz, psize(1)-1.5 )
         ysz = amin1 ( ysz, psize(2)-1.5 )
         call page ( psize(1), psize(2) )
c
c  plot the real(root) vs. x
c
         xsor = ( psize(1)*.5-xsz )*.75
```

```
         if ( xsor.lt..75 )  xsor = ( psize(1)-xsz )*.5
         ysor = ( psize(2)*.5-ysz )*.5
         if ( ysor.lt..5 )  ysor = ( psize(2)-ysz )*.5
         call physor ( xsor, ysor )
c
         call title ( labtre, 16, labx, 16, labwre, 16, xsz, ysz )
         call maptype ( mapset, npoint, xmin, xmax, remin, remax,
     .                  xsz, ysz )
         call dframe
         if ( grids.ne.0 )  call grid ( -grids, -grids )
         if ( mapset.lt.4 )  go to 100
            if ( nore.eq.1 )  go to 120
            call conmak ( rootre, npnew, nrnew, "scale" )
            call contur ( 4, "labels", "draw" )
            go to 110
  100    continue
            call curves ( x, rootre, npoint, -1, nr )
  110    continue
         call endgr ( 0 )
c
c  plot imaginary(root) vs. x
c
  120 continue
         xsor = psize(1)*.5 + ( psize(1)*.5-xsz )*.5
         if ( xsor.lt..5+psize(1)*.5 )  xsor = ( psize(1)-xsz )*.5
         ysor = psize(2)*.5 + ( psize(2)*.5-ysz )*.5
         if (ysor.lt..5+psize(2)*.5 ) ysor = ( psize(2)-ysz )*.5
         call physor ( xsor, ysor )
         if ( xsor.lt.psize(1)*.5 .and. ysor.lt.psize(2)*.5 )
     .         call dendpl ( 0 )
c
         call title ( labtai, 16, labx, 16, labwai, 16, xsz, ysz )
         call maptype ( mapset, npoint, xmin, xmax, aimin, aimax,
     .                  xsz, ysz )
         call dframe
         if ( grids.ne.0 )  call grid ( -grids, -grids )
         if ( mapset.lt.4 )  go to 130
            if ( noai.eq.1 )  go to 140
            call conmak ( rootai, npnew, nrnew, "scale" )
            call contur ( 4, "labels", "draw" )
            go to 140
  130    continue
            call curves ( x, rootai, npoint, -1, nr )
  140    continue
         call endgr ( 0 )
         call dendpl ( 0 )
c
      return
c
 1010 format ( 10a8 )
c
      end
```

```
      subroutine  setfile (names, n)
c
c subroutine setfile appends the current suffix (channel) to
c all the given files.
c
c parameters:
c     names - an one-dimensional integer array with file names
c                 in it.  all'names are limited to 7 characters.
c     n       - an integer that indicates the number of files.
c
c library used:
c     fortlib
c
c
      integer  names(n), n
      integer  userno, account, df, suffix, i
c
c set unit 59 to be the controller or terminal
c
      call msglink (59, 1)
c
c get the suffix
c
      call userinfo (userno, account, df, suffix)
c
c change the left-justified suffix to right-justified
c
      suffix = suffix / 4000000000000000000b
c
c append the suffix to all file names
c
      do 10 i = 1, n
         names(i) = 400b * names(i) + suffix
   10 continue
c
      return
      end
```

```
      real function  stepsz ( vmin, vmax, size, delta )
      real       vmin, vmax, size, delta
c
c  find step size for a graph
c
      delta = amax1 ( 0.001, (vmax-vmin)*.05 )
      stepsz = ( vmax-vmin+2.*delta ) / max0 ( int(size), 4 )
      return
      end
```

```
      subroutine  vvsetup
      complex  c(30), p(1024,100), y(1024)
      integer  l(30)
      common   /parmbk/  l, c, p, y
      data     l(30) /1/
c
      l(30) = l(30) - 1
      return
      end
```